Cycle Basis Algorithms for Reducing Maximum Edge Participation

Fan Wang
Department of Computer Science, University of California, Irvine, CA, USA

Sandy Irani
Department of Computer Science, University of California, Irvine, CA, USA

Abstract

A cycle basis of a graph is a minimal set of cycles from which every cycle in the graph can be generated by symmetric difference. We study the problem of constructing cycle bases of graphs with low maximum edge participation, defined as the maximum number of cycles in the basis that share any single edge. This quantity, though less studied than total weight or length, plays a critical role in quantum fault tolerance, as it directly impacts the overhead of lattice surgery procedures used to implement an almost universal quantum gate set. Building on a recursive algorithm by Freedman and Hastings, we introduce a family of load-aware heuristics that adaptively select vertices and edges to minimize edge participation throughout the cycle basis construction. Our approach improves empirical performance on random regular graphs and on graphs derived from small quantum codes. We further analyze a simplified balls-into-bins process to establish lower bounds on edge participation. While the model differs from the cycle basis algorithm on real graphs, it captures what can be proven for our heuristics without using more complex graph theoretic properties related to the distribution of cycles in the graph. Our analysis suggests that the maximum load of all of our heuristics will be $\Omega(\log^2 n)$. Our results indicate that careful cycle basis construction can yield significant practical benefits in the design of fault-tolerant quantum systems. Maximum edge participation has been studied in the graph theory literature under the name basis number, which is the minimum possible maximum edge participation over all cycle bases in a graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph algorithms, Cycle Basis, Quantum fault tolerance

Digital Object Identifier 10.4230/LIPIcs.SEA.2026.27

Related Version *Previous Version*: `https://arxiv.org/abs/2511.10961`

Supplementary Material *Software (Source code)*: `https://github.com/FanWang000/Cycle_basis_algorithms_public` [31]; archived at `swh:1:dir:ef7c07c921a0cf1b423e9089254a8b82a7780b4c`

1 Introduction

A *cycle basis* of an undirected graph $G = (V, E)$ is a minimal set of cycles such that every cycle in G can be expressed as the symmetric difference (i.e., edgewise XOR) of some subset of the basis cycles (i.e., cycles in the basis). The cycles are not necessarily simple and are just viewed as Eulerian subgraphs, i.e., subgraphs in which every vertex has even degree. For any connected graph with n vertices and m edges, the dimension of the cycle space – and hence the number of basis cycles – is exactly $m - n + 1$ [11].

One intuitive way to see this is via a spanning tree of G: such a tree has exactly $n - 1$ edges and no cycles. Each of the remaining $m - (n - 1) = m - n + 1$ non-tree edges, when added back to the spanning tree, closes a unique simple cycle. These cycles, formed by combining each non-tree edge with the unique tree path connecting its endpoints, constitute a valid cycle basis. A cycle basis constructed this way is called a *fundamental cycle basis*.

Much of the literature on cycle bases focuses on finding a minimum-weight cycle basis in weighted graphs, where the weight of a cycle is the sum of its edge weights, and the weight of the basis is the sum of the weights of all basis cycles [15, 17, 16]. In unweighted graphs,

this reduces to minimizing the total length of the basis cycles. In contrast, we consider a different structural property that disregards edge weights: the *maximum edge participation* of a basis. This measures the maximum number of basis cycles that any edge participates in. Formally, for a cycle basis B of $G = (V, E)$, we define: $\mu(B) = \max_{e \in E} |\{C \in B : e \in C\}|$. The motivation for minimizing the maximum edge participation comes from a well-known construction for quantum fault tolerance and is described further in Section 2.

1.1 Previous Work

The *basis number* for a graph, denoted simply by μ, is the smallest $\mu(B)$ that can be obtained for any cycle basis B [28].

For certain families of graphs, tight bounds on the basis number μ are known. Planar graphs satisfy $\mu \leq 2$ [22]. Any complete graph with $n \geq 5$ has $\mu = 3$, and this characterization is tight: for any $n \geq 5$, if a graph has $\mu = 3$, then it must be complete [28]. More recently, the basis number has been related to topological properties of the graph: if a graph can be embedded on a surface of genus g, then $\mu = O(\log^2 g)$ [18]. Since any graph with m edges can be embedded on a surface of genus $O(m)$ [23], this implies the general bound $\mu = O(\log^2 m)$. In this paper, we restrict attention to simple graphs or multigraphs with constant edge multiplicity (as is sufficient for the applications in Section 2), yielding $\mu = O(\log^2 n)$.

On the other hand, a general lower bound holds for any connected graph: $\mu \geq \text{girth}(G)\,(m - n + 1)/m$, as shown in [1].

For some constant-degree expander families such as *LPS expanders* [21], which have logarithmic girth, this implies $\mu = \Omega(\log n)$.

1.2 The Freedman–Hastings algorithm

While theoretical bounds on $\mu(B)$ are known for some graph families, few algorithmic approaches explicitly target minimizing this quantity. A notable exception is the recent recursive algorithm of Freedman and Hastings [8], which on any constant-degree graph constructs a cycle basis B, such that with high probability $\mu(B) = O(\log^2 n)$. The cycle basis B produced by the Freedman-Hastings algorithm produces a cycle basis consisting of simple cycles that is also *weakly fundamental*. A cycle basis is called *weakly fundamental* if its cycles can be placed in a linear ordering such that each cycle contains at least one edge that does not appear in any later cycle in the ordering [27]. Their method proceeds recursively, handling one of three cases at each step:

- **Case 1**: If there is a degree-one vertex, remove it.
- **Case 2**: If there is a degree-two vertex v with neighbors x and y: **Case 2A**: If x and y are not connected by an edge, remove v and add an edge between x and y. **Case 2B**: If the edge (x, y) already exists (i.e., the three vertices form a triangle), add the triangle $[x, v, y]$ to the cycle basis and remove v. These two subcases are illustrated in Figure 1.
- **Case 3**: If all vertices have degree at least three, find a short cycle of length $O(\log n)$, add it to the basis, and randomly remove one of its edges from the graph.

In the third case, a short cycle of length $O(\log n)$ is found using a breadth-first search (BFS). In this context, cross edges are defined as edges connecting a newly discovered vertex to an already visited one – i.e., all non-tree edges in the BFS tree. Starting from any vertex, any cross edge encountered during BFS, together with the paths from its endpoints to their lowest common ancestor, forms a cycle. The cycle formed by the first cross edge in the BFS tree is guaranteed to have length at most $2\lceil \log n \rceil$ (see Lemma 4.3 of [16]). In the *Freedman-Hastings* algorithm, the BFS root is chosen uniformly at random, and they use the cycle generated by the first cross edge encountered.

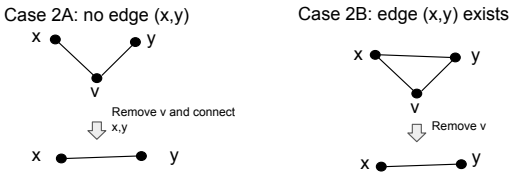

Figure 1 Illustration of **Case 2A** and **Case 2B** in the *Freedman–Hastings* cycle basis algorithm. Vertex v has degree two, with neighbors x and y. In **Case 2A**, we proceed recursively on the updated graph and after computing a cycle basis, reinsert v if any cycle basis contains (x, y); in **Case 2B**, we add the triangle $[x, v, y]$ to the cycle basis and proceed recursively on the updated graph.

As noted in [8], the algorithm can be extended to multigraphs via a preprocessing step: all self-loops and parallel edges are first removed, and the cycles induced by these edges are added directly to the cycle basis.

However, the $O(\log^2 n)$ bound achieved by this algorithm may be pessimistic for typical graphs. In [32], it was conjectured that for constant-degree random expanders, it might be possible to construct a cycle basis such that the expected maximum edge participation is only $O(\log n)$. From this perspective, the *Freedman–Hastings* construction can be seen as a worst-case bound, while more efficient average-case constructions may be possible with more refined cycle selection strategies. More broadly, the problem of minimizing maximum edge participation over all cycle bases remains a largely open combinatorial question, with practical implications in contexts such as quantum error correction and fault-tolerance [13, 12].

1.3 Our contribution

In the *Freedman–Hastings* algorithm, several design choices remain flexible and open to optimization, including:

- the starting vertex (root) for BFS;
- the selection of the cross edge used to extract a short cycle;
- the choice of which edge to remove from the cycle.

We explore heuristics that refine the edge and vertex selection rules, proposing a load-aware variant of the algorithm. The *load* of an edge is defined as the number of basis cycles it has appeared in so far, while the *load* of a vertex is the average load of its incident edges. To better reflect structural changes, we further define how load evolves during transformations: in Case 2A (where a new edge is inserted between two neighbors of a removed vertex), the load of the new edge is set to the maximum of the loads of the two removed edges. In Case 2B (where the connecting edge already exists), we increment the load of that existing edge by one. Among the variants we tested, the following heuristic performed best empirically on random regular graphs:

- initialize each BFS from the vertex with the highest load;
- select the first cross edge that forms a cycle containing the BFS root;
- remove the *highest-load* edge from the resulting cycle.

In Section 2 we give an overview of how the cycle basis problem arises in quantum fault tolerance. In Section 3, we present experimental results on random regular graphs to compare several of our best-performing heuristics. Section 4 applies our algorithm to graphs derived from small quantum LDPC codes, demonstrating its practical relevance. The implementation

and experimental scripts are available as additional material [31]. Finally, in Section 5, we define a random process based on a balls-and-bins model to analyze and minimize the maximum load on a bin. This process serves as a proxy for the cycle basis problem on a random graph, capturing the distribution of load on edges under the simplifying assumption that cycles are random sets of edges. The edges chosen in a cycle is modeled by a random set of edges. The goal is to capture the best analysis that can be achieved without understanding the more complex distribution of edges chosen when cycles are selected in a real graph. Our analysis suggests that the maximum load of all of our heuristics will be $\Omega(\log^2 n)$.

2 Motivation from Quantum Fault Tolerance

Due to the fragile nature of quantum systems, quantum error correction (QEC) is an important challenge for building quantum computers capable of practical applications. QEC protects quantum information from environmental noise by redundantly encoding it using a quantum error-correcting code (QECC). In recent years, a particular class of QECCs known as *quantum low-density parity-check* (QLDPC) codes [10] has attracted significant attention for their potential to reduce the overhead associated with encoding and logical operations. A parity check is a measurement which measures whether an error has occurred. Low-density parity checks, which only operate on a constant number of qubits, are easier to implement in a fault-tolerant way. Recent works [24, 5, 25, 19] have proposed constructions of *good* QLDPC codes – those with constant rate and linear distance.

However, a major challenge remains: how to perform logical gates fault-tolerantly on information encoded using these codes. A technique known as *lattice surgery* was originally developed for implementing logical gates on surface codes [14], and has since been generalized to broader classes of QLDPC codes [6, 7, 32, 13]. The core goal of lattice surgery is to enable fault-tolerant measurements of logical operators, which in turn allows the implementation of an almost universal gate set – the Clifford group. This set, when supplemented by *magic state distillation* [4], yields a universal gate set for quantum computation [20].

Every QLDPC code has a Tanner graph representation, where physical qubits and checks are nodes, and edges indicate which checks act on which qubits. We define the *weight* of a check to be the number of physical qubits it touches, and the *degree* of a qubit to be the number of checks connected to it.

Without loss of generality, we focus on CSS codes, which involve only two types of checks: Z-checks and X-checks. (In general, a check can involve both X and Z operators.) These two types of checks correspond to the two common types of errors in quantum systems: X-errors (bit-flip errors) and Z-errors (phase-flip errors). For CSS codes, the Tanner graph is a tripartite graph – one part for qubits, one for X-checks, and one for Z-checks – analogous to the bipartite structure in classical LDPC codes.

Correspondingly, there are two types of logical measurements: Pauli X measurements and Pauli Z measurements. In this discussion, we focus on Pauli X measurements; the procedure for Pauli Z measurements is analogous. A Pauli X measurement can be viewed as a measurement on a subset of physical qubits. However, such measurements often have high weight, meaning they involve many physical qubits. Performing high-weight measurements directly is challenging in a fault-tolerant manner.

The goal of lattice surgery is to decompose such high-weight measurements into a sequence of lower-weight, fault-tolerant measurements. The basic idea is to introduce an ancillary system (which can also be thought of as another QLDPC code) that is coupled to the original

code on which we wish to perform the measurement. By measuring all the X-checks in the ancilla system, one can effectively obtain the outcome of the desired high-weight Pauli X measurement.

There have been several approaches for constructing the ancilla system [6, 7, 32]. These constructions share a common starting point: they begin with the Tanner graph and focus on the induced subgraph of the support (i.e., the set of qubits involved in the Pauli X measurement.) The cycle basis problem comes out of the Williamson et. al. construction. Although the details of the construction are beyond what we can cover here, the result of the construction is a new graph based on the induced subgraph of the Tanner graph in which edges represent physical qubits, vertices correspond to X-checks, and Z-checks are defined via a cycle basis B of the graph G, which together define the ancilla system.

Each basis cycle corresponds to a Z-check in the ancilla system, and its length determines the weight of that check. Long cycles lead to high-weight checks, which can violate the LDPC property of the original code. However, there is a straightforward method to resolve this: cellulation – the process of subdividing long cycles into multiple short cycles by adding auxiliary edges, thereby restoring LDPC properties.

In contrast, maximum edge participation corresponds to the maximum qubit degree in the ancilla system – that is, the number of Z-checks attached to a given qubit. High qubit degrees can violate the LDPC property of the code, and unlike the case of high-weight checks, there is no comparably low-cost fix such as cellulation. The overhead of the construction is then dominated by the cost of reducing the maximum edge participation.

3 Experimental Comparison of Different Heuristics on Random Regular Graphs

In this section, we present five different variants of the original Freedman–Hastings algorithm by varying the three key design choices listed in Table 1. We will refer to the Freedman–Hastings algorithm as Version 0.

Table 1 Variants of the Freedman–Hastings algorithm with different design choices (Version 0 is the original version).

Variant Name	BFS Root Selection	Cross Edge Selection	Edge Removal Strategy
Version 0	Random vertex	First encountered	Random edge
Version 1	Random vertex	First encountered	Max-load edge
Version 2	Max-load vertex	First encountered	Max-load edge
Version 3	Max-load vertex	First forming cycle with root	Max-load edge (prefer incident to root)
Version 4	Max-load vertex	First forming cycle with root	Load-weighted probabilistic removal
Version 5	Each vertex	Shortest cycle over all roots	Max-loaded edge

The only difference between Version 1 and Version 0 is the edge removal strategy: while Version 0 removes a random edge from the selected cycle, Version 1 removes the edge with the highest load (breaking ties randomly). This strategy helps reduce long-term edge congestion by preferentially eliminating the most heavily used edges early in the recursion.

The difference between Version 2 and Version 1 lies in the BFS root selection. While Version 1 selects the BFS root randomly, Version 2 always starts from the vertex with the highest load. Although this factor may seem less impactful than edge removal, empirical results show that prioritizing the max-load vertex leads to better performance. Intuitively, a high-load vertex is likely incident to multiple heavily loaded edges. By starting BFS from such a vertex, the resulting cycle has a higher chance of containing one of these high-load edges, which can then be removed using the same greedy strategy.

Version 3 improves upon Version 2 by refining both the cross edge selection and edge removal strategies. Instead of selecting the first encountered cross edge during BFS, we restrict the choice to those that yield a cycle containing the BFS root. This ensures that the starting vertex – which is chosen based on maximum load – remains in the cycle, increasing the chance of targeting heavily used edges.

Additionally, when multiple edges in the cycle have the same maximum load, we prioritize removing an edge that is incident to the BFS root. This tie-breaking rule is particularly effective in the later stages of the algorithm when the graph becomes sparse. For instance, in a 3-regular graph, removing one edge via a short cycle (as in Case 3) typically creates two degree-two vertices – the endpoints of the removed edge – which will each trigger additional reductions via Case 2; see Figure 2 for an illustrative example. If the BFS root lies on the cycle and has high load, this refinement increases the likelihood of further removing overloaded edges connected to it, thus improving overall balance in the cycle basis.

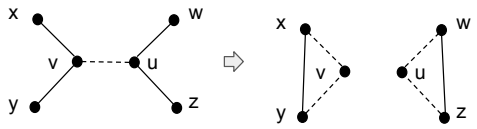

Figure 2 Illustration of how Case 2 introduces additional reductions in a 3-regular graph following Case 3. Suppose edge (u, v) is removed as part of a short cycle in Case 3. After removal, both u and v become degree-two vertices. By Case 2A, edges (x, v) and (v, y) are replaced by (x, y), and the load on (x, y) is set to the maximum of the loads on (x, v) and (v, y). This gives an advantage only if all 3 edges incident to v have high load. A similar reduction applies to the pair (w, z) resulting from the vertex u.

Version 4 introduces a probabilistic edge removal strategy. Instead of deterministically removing the highest-load edge, we remove edge i from the selected cycle C with probability $2^{L(i)} / \sum_{j \in C} 2^{L(j)}$, where $L(i)$ denotes the load of edge i. This softmax-style weighting introduces randomness into the algorithm, which may help avoid worst-case scenarios that can trap greedy variants. However, empirical results suggest that Version 4 does not outperform Version 3 on average.

Finally, Version 5 is a significantly more expensive algorithm. It selects the cycle of minimum length and then removes the maximum-load edge from that cycle. To identify the shortest cycle, the algorithm performs a BFS from each vertex as the root, searches for a cycle containing that root, and then takes the shortest cycle among all roots. Owing to its high computational cost, we evaluate Version 5 only in the next section, where the graphs – arising from small quantum codes – are relatively small.

We evaluate the variants (Versions 0–4) on random regular graphs, which are known to be asymptotically close to Ramanujan graphs – optimal spectral expanders – according to the result of Friedman [9]. This motivates our choice, as most random d-regular graphs are constant-degree expanders, resembling the structure of graphs that arise in real quantum LDPC codes.

To generate these graphs, we use Python's NetworkX package, which internally implements the configuration model of Steger and Wormald [30]. Specifically, we test our heuristics on random 3-regular and 8-regular graphs across a range of sizes. We chose these graphs to roughly mimic the structure of near-term quantum codes, which typically do not exhibit very high connectivity, check weight, or qubit degree. For each graph size, we generate a number of random instances (as detailed in Table 2) and apply all five algorithmic variants.

Table 2 Number of random d-regular graphs generated per size n, for each degree d.

Degree / Graph size n	32	64	128	256	512	1024	2048	4096	8192	16384
$d = 3$	1600	1600	1600	1600	1600	800	400	200	100	50
$d = 8$	1600	1600	1600	1600	1600	800	400	200	100	–

Note: No experiments were conducted for $d = 8$ at $n = 16384$ due to runtime constraints. Completing all trials required approximately 42 hours.

Implementation details. A naïve recursive implementation would repeatedly scan all vertices to identify degree–1 or degree–2 vertices and to select a start vertex for cycle search, incurring $\Theta(n)$ work per recursion step. Since each step removes or contracts only a constant number of vertices or edges, this leads to an overall $\Theta(mn)$ overhead even before accounting for the cost of cycle searches.

To avoid this inefficiency, we maintain two *queues* containing vertices that may currently have degree 1 or 2. After each local modification of the graph, only the affected neighboring vertices are re-checked and (if applicable) enqueued. Queue entries may become *stale*, meaning that they no longer reflect the current graph state (e.g., the vertex has already been removed or its degree has changed); such entries are detected and discarded upon extraction. This approach yields amortized $O(1)$ work per update for degree tracking.

To guide cycle selection, we additionally maintain a priority queue over vertices keyed by their current load. Whenever a vertex's load changes, a new entry is inserted into the heap, and outdated entries are discarded lazily when extracting the maximum-load vertex. Each load update therefore incurs amortized $O(\log n)$ overhead. As a result, all global scans over vertices are eliminated, and the overhead outside the BFS-based cycle search is reduced from $\Theta(n)$ per recursion step to amortized $O(\log^2 n)$ (since each recursion may trigger $O(\log n)$ load updates), which is essential for scalability to large graphs.

Running time analysis. All variants (Versions 1–4) have the same worst-case asymptotic running time as the original Version 0 (the *Freedman–Hastings* algorithm). Each recursive step removes at least one edge and may invoke a breadth-first search (BFS) to identify a cycle, which takes $O(m)$ time in the worst case. Since at most $O(m)$ such steps can occur over the entire recursion, the overall worst-case running time remains $O(m^2)$ (equivalently, $O(n^2)$ for constant-degree graphs).

Our implementations primarily reduce constant factors by eliminating repeated global scans over all vertices. In particular, degree–1 and degree–2 vertices are maintained using queues, while candidate start vertices for cycle search are selected via a priority queue keyed by vertex load. As a result, degree updates and load updates incur only local $O(1)$ or amortized $O(\log^2 n)$ work per recursion step, rather than $\Theta(n)$-time global scans. Although these optimizations do not improve the worst-case asymptotic bound, they substantially reduce the overhead outside of the BFS-based cycle search and are crucial for scaling the algorithms to larger graphs in practice.

As graph sizes grow exponentially as in Table 2, we correspondingly reduce the number of trials for larger instances to keep the total computation time manageable. All experiments were performed on a desktop workstation (Intel® Core™ i7–14700F CPU, 16 GB RAM, 64-bit Windows 11), and completing all trials required approximately 42 hours.

Results and Visualization. We collectively visualize the results across all graph sizes and degrees using the boxplot from multiple trials, as shown in Figure 3. Empirically, all of our algorithmic variants demonstrate significant improvement over the original *Freedman–Hastings* algorithm, suggesting that the max-load edge removal strategy is the primary factor driving these performance gains.

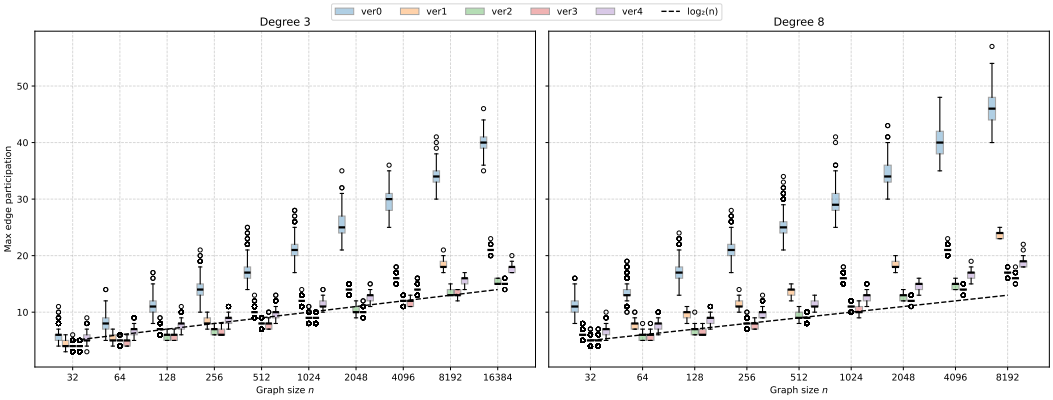

Figure 3 Boxplot comparison of the maximum edge participation across five algorithmic variants of the Freedman–Hastings algorithm, evaluated on random d-regular graphs. The left panel corresponds to $d = 3$, and the right panel to $d = 8$. The boxplot aggregates the results from thousands of random graph instances (see Table 2). Each variant is color-coded, and the black dashed curve in both panels shows the baseline $\log_2(n)$ scaling for reference.

Among the variants, Version 3 consistently achieves the lowest maximum edge participation on average, followed by Version 2, then Version 4, and finally Version 1. This ordering confirms that both the BFS root selection and cross edge selection strategies contribute meaningfully to reducing edge congestion – beyond the effect of edge removal alone.

To further understand the asymptotic behavior of the different algorithmic variants, we plot in Figure 4 the ratio between the actual median maximum edge participation and fitted functions of the form $c \log_2(n)$ and $c \log_2^2(n)$, where the coefficient c is independently optimized for each curve. This normalization removes the leading-order growth rate and highlights the deviation from ideal logarithmic or polylogarithmic scaling.

We include results for Version 0 and Version 3 of the algorithm across both degree-3 and degree-8 random regular graphs. All curves use solid lines, with different marker shapes distinguishing the vertex degree. In the left plot, we observe that the ratios for both Version 0 and Version 3 trend upward, indicating super-logarithmic scaling for both versions. In contrast, the right plot shows that the ratios for Version 0 remain relatively flat, whereas those for Version 3 initially decrease but eventually level off. This suggests that Version 3 offers only a constant-factor improvement in scaling relative to Version 0, rather than a fundamentally different asymptotic behavior. This observation is also consistent with the lower bound established for the balls-into-bins proxy in Section 5.

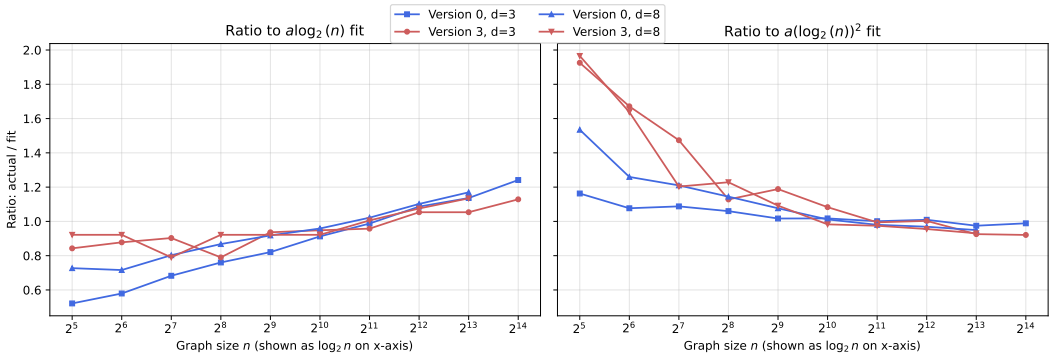

Figure 4 Comparison of the ratio between the actual median maximum edge participation and fitted functions $a\log_2(n)$ (left) and $a\log_2^2(n)$ (right), for Versions 0 and 3 of the algorithm. Each curve is normalized by its own fitted coefficient c, independently optimized per curve. Solid lines are used throughout, with marker shape distinguishing the degree: squares (degree 3) and triangles (degree 8) for Version 0 (blue), and circles (degree 3) and inverted triangles (degree 8) for Version 3 (red). A flat curve near 1 indicates better scaling agreement with the fitted asymptotic model.

4 Experiment on Graphs from Small Quantum Codes

In this section, we evaluate our algorithms on graphs arising from practical quantum codes. Here we consider two different families of codes: quantum radial codes [29] and quantum Tanner Codes [19, 26].

Quantum radial codes were introduced by Scruby et al. [29], which are based on the lifted product construction [24] applied to certain quasi-cyclic classical codes. The lifted product is one of the first constructions known to yield good quantum LDPC codes, and it is also effective for generating near-term deployable codes. Quantum radial codes exhibit promising features for fault-tolerant quantum computing. In particular, they offer comparable error suppression to surface codes while requiring roughly five times fewer physical qubits. Moreover, they have been observed to support single-shot decoding [2], making them strong candidates for near-term implementations.

Quantum Tanner codes were introduced by Leverrier and Zémor [19] as another construction of asymptotically good quantum LDPC codes. Building on this framework, Radebold, Bartlett, and Doherty presented explicit instances of quantum Tanner codes based on dihedral groups and pairs of classical component codes [26]. Numerical simulations show that these explicit codes achieve logical error rates comparable to surface codes of similar distance under both phenomenological and circuit-level noise models, highlighting their potential for fault-tolerant quantum error correction.

4.1 Experiments for Graphs from Quantum Radial Codes

We test our algorithm on graphs arising from two specific quantum radial codes presented in [29]: a $[[90, 8, 10]]$ code and a $[[352, 18, 20]]$ code. Here, a $[[n, k, d]]$ code refers to a quantum code with n physical qubits, k logical qubits, and code distance d. These codes have parameters comparable to those of the Gross code and the Double Gross code [3], but have received relatively less attention in the literature. In [29], the authors provide a canonical basis for all Pauli-X and Pauli-Z logical operators. In our experiments, we focus on the Pauli-X basis (note that the procedure for Pauli-Z measurements would be essentially identical). Following the logical operator measurement construction in [32], we extract, for

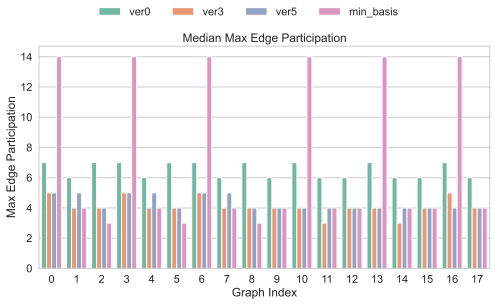

Figure 5 Median maximum edge participation across algorithmic variants for the graphs arising from the $[[352, 18, 20]]$ code.

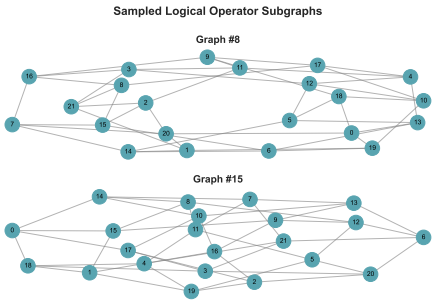

Figure 6 Two example logical-operator–induced subgraphs from the dataset. Each edge represents a qubit in the ancilla system used for lattice surgery, and each vertex corresponds to an X-check. The choice of cycle basis determines the associated set of Z-checks.

each Pauli-X basis operator, an induced subgraph from the Tanner graph representation of the code. Specifically, each intersecting Z-check becomes an edge in the induced graph; in the two quantum radial codes we study, every Z-check intersects each logical operator on exactly two qubits, so this conversion is unambiguous.

For the $[[90, 8, 10]]$ code, this yields 8 graphs (one for each logical X operator), and for the $[[352, 18, 20]]$ code, we obtain 18 graphs. In general, to ensure distance preservation during lattice surgery, one may need to augment these graphs with additional edges to boost their Cheeger constants to at least 1, as this is a sufficient (but not necessary) condition for maintaining code distance. However, in our case, we observe that for the smaller code, all 8 graphs already have Cheeger constant exactly 1. For the larger code, 12 out of the 18 graphs have Cheeger constant ≥ 1, 3 have approximately 0.91, and the remaining 3 have values near 0.73. Given these relatively high expansion properties, we directly test our algorithms on these induced subgraphs without further augmentation, expecting the results to closely reflect those obtained in a full lattice surgery simulation.

The algorithms we test include Version 0 (the original Freedman–Hastings algorithm), Version 3 (our best-performing heuristic on random regular graphs), Version 5 (the variant that always selects the shortest cycle at each recursion), and min-basis (a minimum-weight cycle basis algorithm [17]). We include Version 5 and the minimum-weight cycle basis because, in the context of lattice surgery, shorter basis cycles may reduce overhead, even though maximum edge participation is typically the dominant cost.

We first focus on the induced graphs arising from the larger code. These graphs each have 22 vertices and are 4-regular; two representative examples are shown in Figure 6. The corresponding performance comparison is presented in Figure 5. For each such graph, we evaluate each algorithm over 200 independent runs and report the median maximum edge participation.

As shown in Figure 5, Version 3 consistently achieves the best performance on these graphs, yielding roughly a one-third reduction in maximum edge participation compared to Version 0. The min-basis algorithm can occasionally match the performance of Version 3, but also exhibits substantial variability and can produce the worst outcomes, which is expected since it optimizes cycle length rather than edge participation. Version 5 most often matches the performance of Version 3 and consistently improves over Version 0, although on some instances it is slightly worse than Version 3.

We briefly comment on the induced graphs arising from the smaller code. These graphs have 10 vertices – comparable to the code distance – and are 3-regular. Representative examples and the corresponding performance plots are included in the additional material [31]. For these smaller graphs, each algorithm is run 100 times per graph. The observed performance differences among the algorithms are relatively minor, reflecting the limited graph size and degree.

4.2 Experiments for Graphs from Quantum Tanner Codes

Similar to the case of quantum radial codes, we evaluate our algorithms on graphs arising from two quantum Tanner codes introduced in [26]: a $[[72, 14, 4]]$ code and a $[[250, 10, 15]]$ code. Unlike the previous setting, the graphs induced by these Tanner codes are generally irregular and span a wider range of sizes. For the $[[72, 14, 4]]$ code, the 14 induced graphs have sizes ranging from 8 to 14 vertices, with maximum degrees between 2 and 4. For the $[[250, 10, 15]]$ code, the 10 induced graphs have sizes ranging from 20 to 55 vertices, with maximum degrees between 5 and 7.

The experimental results for the larger Tanner code are deferred to Appendix B. In particular, the performance results are shown in Figure 9, with two representative induced subgraphs illustrated in Figure 10. The corresponding performance results for the smaller Tanner code, together with representative induced subgraphs, are provided in the additional material [31]. For each graph of the larger (respectively, smaller) code, we run each algorithm 200 (respectively, 100) times and report the median maximum edge participation.

Overall, the results are consistent with those observed for random regular graphs and graphs derived from quantum radial codes. In particular, Version 3 continues to perform robustly across all tested instances, typically achieving the lowest or near-lowest maximum edge participation. Version 5 often matches the performance of Version 3 but occasionally performs slightly worse, while incurring a higher computational cost due to its exhaustive shortest-cycle search. The min-basis algorithm again exhibits substantial variability: although it can sometimes achieve competitive performance, it can also produce the worst outcomes, reflecting its optimization for cycle length rather than edge participation.

These experiments demonstrate that our load-aware heuristics – and Version 3 in particular – remain effective even for irregular graphs with heterogeneous degree distributions and sizes, indicating that their benefits extend beyond the regular graph families considered earlier.

5 A balls-into-bins model as a proxy

We consider a balls-into-bins model as a proxy for the behavior of our algorithms on 3-regular graphs. In this model, each bin corresponds to an edge in the graph, and each selected cycle is a subset of k bins. The load on each edge is represented by placing a ball in each of the selected bins. Then three bins are removed, representing the edges that are removed from the graph. Note that when the graph is exactly 3-regular, each iteration of Case 3 (followed by two applications of Case 2A) decreases the total number of edges by three, as illustrated in Figure 2. Indeed, the local transformation removes five existing edges and inserts two new edges, so the net change in the number of edges is -3.

The goal of the model is to understand the extent to which we can analyze the asymptotic behavior of our best variant without using detailed graph-theoretic properties. As discussed above, one iteration consisting of Case 3 followed by two applications of Case 2A can potentially remove up to three heavily loaded edges from the graph. In the balls-to-bins model, we model the selection of a cycle as selecting the three most heavily loaded edges/bins,

Algorithm 1 Comparison of two processes.

Process 1

Require: $M > 0$, the initial number of buckets.
$m \leftarrow M$
$n \leftarrow \frac{2}{3}M$
All m buckets start empty.
$m_{\min}$ is some fixed constant.
while $m > m_{\min}$ **do**
  $k \leftarrow 2\lceil \log_2 n \rceil$
  Select the 3 buckets with the highest load.
  Select $k-3$ additional buckets at random.
  Add one ball to each of the k selected buckets.
  Remove the three most heavily loaded buckets.
  $m \leftarrow m - 3$
  $n \leftarrow \frac{2}{3}m$
end while

Process 2

Require: $M > 0$, the initial number of buckets.
$m \leftarrow M$
$j \leftarrow 1$
All m buckets start empty.
$m_{\min}$ is some fixed constant.
while $m > m_{\min}$ **do**
  $m_{old} \leftarrow m$
  $n \leftarrow \frac{2}{3}m$.
  $k \leftarrow 2\lceil \log_2 n \rceil$
  while $m > \frac{M}{2^j}$ **do**
    Select k buckets at random.
    Add one ball to each of the k selected buckets.
    $m \leftarrow m - 3$
  end while
  $r \leftarrow (m_{old} - m)$
  Remove the r most heavily loaded buckets.
  $j \leftarrow j + 1$
end while

and the rest of the $k-3$ bins are selected at random. Then the three most heavily loaded bins are removed from the system, representing the removal of those edges from the graph. This model is favorable to the algorithm by allowing the algorithm to select and remove the three most heavily loaded bins in the entire system. In a real graph, the three most heavily loaded edges might not even be located on a single short cycle. The selection of the rest of the edges is simplified to be a random set. Intuitively, getting a better analysis for the behavior of our cycle basis algorithms will require a more sophisticated understanding of the distribution of the edges participating in each selected cycle.

There are a few other ways in which our proxy process differs from cycle basis selection in a real graph. The proxy process assumes that the cycle selected will have length exactly $k = 2\lceil \log_2 n \rceil$ in each round. The versions which select the cycle arising from the first cross edge encountered may yield shorter cycles, resulting in less load on the edges. To better justify this choice, we provide an analysis of the cycle length produced during a BFS exploration of a random regular graph. In particular, we show that with constant probability, the exploration remains tree-like up to depth $\Theta(\log n)$, and the first back edge creates a cycle of length $\Omega(\log n)$. The detailed proof is deferred to Appendix E. Thus, without a more refined model for how the graph evolves over time, we cannot exploit the possibility of shorter selected cycles to obtain sharper bounds on the performance of our heuristics. The proxy process also does not capture vertex removals from Cases 1 and $2B$. However, our numerical results suggest that these are relatively infrequent events and therefore do not contribute significantly to edge load as shown in Appendix A.

The balls-and-bins process is called *Process 1* and is given in Algorithm 1. However, Process 1 remains analytically challenging. To facilitate analysis, we define a further simplification, which still preserves the essential features needed for bounding load growth.

In the simplified process, we define an *epoch* as a sequence of iterations in which the number of buckets m decreases by a factor of 2. During each iteration within an epoch, we add a ball to each of k randomly chosen bins and decrease m, the counter for the number of buckets, by 3. However, the buckets are not actually removed until the end of an epoch, at which point a new epoch begins. The delayed-removal process is called *Process 2* and is given in Algorithm 1.

This delayed-deletion model allows additional load to accumulate before pruning. As a result, more balls are discarded during each epoch compared to Process 1. Therefore, any lower bound established for Process 2 also applies to Process 1, and thus to the original algorithm. In Appendix D, we provide a formal proof that Process 2 lower-bounds Process 1 if we select $(k-3)$ buckets instead of k buckets in Process 2; note that this modification does not affect the asymptotic lower bound for Process 2 discussed below.

In what follows, we prove the following theorem, which provides a lower bound on the maximum load $L_{\max}$ in Process 2. All logarithms are base 2, unless stated otherwise.

▶ **Theorem 1.** *For any fixed $c \in (0, 0.16)$, process 2 satisfies*

$$L_{\max} \;=\; \Omega\big(\log^2 M\big) \quad \textit{with high probability.}$$

Explicitly, $L_{\max} \;\geq\; \frac{c^2}{6}\left(1-\frac{c}{2}\right)\log^2 M \quad$ *w.h.p.*

Before proving the theorem, we prove several lemmas. Define $m_j = M/2^j$, $n_j = \frac{2}{3}m_j$, and $k_j = 2\lceil \log_2 n_j \rceil$. During epoch j, we begin with m_j buckets and perform ball tossing for $m_j/6$ rounds. In each round, we toss k_j balls into k_j distinct buckets chosen uniformly at random *without replacement*. No deletions occur during the rounds.

Consider a particular an epoch j. The probability that a fixed bucket is chosen during a single round in epoch j is k_j/m_j. The incremental load placed on any bucket during epoch j is

$$X_j \;\sim\; \text{Binomial}\Big(\tfrac{m_j}{6}, \tfrac{k_j}{m_j}\Big), \mu_j := \mathbb{E}[X_j] = \frac{m_j}{6}\,\frac{k_j}{m_j} = \frac{k_j}{6}.$$

Fix a constant $c \in (0, 0.16)$ and call a bucket *bad in epoch j* if the incremental load satisfies $X_j \leq c\,\mu_j$. During epoch j let B_j be the number of *bad* buckets. We state the following two lemmas, whose proofs are provided in Appendix C.

▶ **Lemma 2** (Few bad buckets in one epoch)**.** *Fix $c \in (0, 0.16)$ and set $\alpha = \frac{(1-c)^2}{6\ln 2}$. During epoch j, let B_j be the number of buckets whose incremental load is at most $c\mu_j$. Then*

$$\Pr\left(B_j \geq \left(\tfrac{3}{2}\right)^{\alpha} m_j^{1-\frac{\alpha}{2}}\right) \leq m_j^{-\frac{\alpha}{2}}.$$

▶ **Lemma 3** (Few bad buckets over L epochs)**.** *Fix $c \in (0, 0.16)$ and define $\alpha = \frac{(1-c)^2}{6\ln 2}$. Let $L = \left\lceil \frac{c}{2}\log_2 M \right\rceil$. Then with probability $1 - o(1)$, the total number of buckets that are bad in* any *of the first L epochs is at most*

$$C(c)\, M^{1-\frac{\alpha}{2}}, \qquad \textit{where} \qquad C(c) := \left(\tfrac{3}{2}\right)^{\alpha} \cdot \frac{1}{1-2^{-(1-\frac{\alpha}{2})}}.$$

We are now ready to prove Theorem 1. The basic idea is that for the chosen value of L (which is $\Omega(\log n)$), with high probability the number of buckets that have ever gone bad in L rounds is less than the total number of buckets remaining. Thus, with high probability, there will be a bucket remaining which was never bad in the first L rounds. A bucket with this property received $\Omega(\log n)$ incremental load in each of the L rounds, resulting in a bucket with total load $\Omega(\log^2 n)$. For a complete proof of Theorem 1 check Appendix C.

References

1 John Anthony Banks and Edward F Schmeichel. The basis number of the n-cube. *Journal of Combinatorial Theory, Series B*, 33(2):95–100, 1982.

2 Héctor Bombín. Single-shot fault-tolerant quantum error correction. *Physical Review X*, 5(3):031043, 2015.

3 Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024. doi:10.1038/S41586-024-07107-7.

4 Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A—Atomic, Molecular, and Optical Physics*, 71(2):022316, 2005.

5 Nikolas P Breuckmann and Jens N Eberhardt. Balanced product quantum codes. *IEEE Transactions on Information Theory*, 67(10):6653–6674, 2021. doi:10.1109/TIT.2021.3097347.

6 Lawrence Z Cohen, Isaac H Kim, Stephen D Bartlett, and Benjamin J Brown. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Science Advances*, 8(20):eabn1717, 2022.

7 Andrew Cross, Zhiyang He, Patrick Rall, and Theodore Yoder. Linear-size ancilla systems for logical measurements in qldpc codes. *arXiv e-prints*, pages arXiv–2407, 2024.

8 Michael Freedman and Matthew Hastings. Building manifolds from quantum codes. *Geometric and Functional Analysis*, 31(4):855–894, 2021.

9 Joel Friedman. A proof of alon's second eigenvalue conjecture. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 720–724, 2003. doi:10.1145/780542.780646.

10 Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *arXiv preprint*, 2013. arXiv:1310.2984.

11 Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

12 Matthew B Hastings. On quantum weight reduction. *arXiv preprint*, 2021. arXiv:2102.10030.

13 Zhiyang He, Alexander Cowtan, Dominic J Williamson, and Theodore J Yoder. Extractors: Qldpc architectures for efficient pauli-based computation. *arXiv preprint*, 2025. arXiv:2503.10390.

14 Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

15 Joseph Douglas Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing*, 16(2):358–366, 1987. doi:10.1137/0216026.

16 Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina A Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199–243, 2009. doi:10.1016/J.COSREV.2009.08.001.

17 Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna E Paluch. An algorithm for minimum cycle basis of graphs. *Algorithmica*, 52(3):333–349, 2008.

18 Florian Lehner and Babak Miraftab. Basis number of bounded genus graphs. *arXiv preprint*, 2024. arXiv:2410.10566, doi:10.48550/arXiv.2410.10566.

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. doi:10.1109/FOCS54457.2022.00117.

20 Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019. doi:10.22331/Q-2019-03-05-128.

21 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. doi:10.1007/BF02126799.

22 Saunders Mac Lane. *A combinatorial condition for planar graphs*. Seminarium Matemat., 1936.

23 Martin Milgram and Peter Ungar. Bounds for the genus of graphs with given betti number. *Journal of Combinatorial Theory, Series B*, 23(2-3):227–233, 1977. doi:10.1016/0095-8956(77)90033-8.

24 Pavel Panteleev and Gleb Kalachev. Quantum ldpc codes with almost linear minimum distance. *IEEE Transactions on Information Theory*, 68(1):213–229, 2021. doi:10.1109/TIT.2021.3119384.

25 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. doi:10.1145/3519935.3520017.

26 Rebecca Katharina Radebold, Stephen D Bartlett, and Andrew C Doherty. Explicit instances of quantum tanner codes. *arXiv preprint*, 2025. arXiv:2508.05095.

27 Romeo Rizzi. Minimum weakly fundamental cycle bases are hard to find. *Algorithmica*, 53(3):402–424, 2009. doi:10.1007/S00453-007-9112-8.

28 Edward F Schmeichel. The basis number of a graph. *Journal of Combinatorial Theory, Series B*, 30(2):123–129, 1981. doi:10.1016/0095-8956(81)90057-5.

29 Thomas R Scruby, Timo Hillmann, and Joschka Roffe. High-threshold, low-overhead and single-shot decodable fault-tolerant quantum memory. *arXiv preprint*, 2024. arXiv:2406.14445.

30 Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999. URL: http://journals.cambridge.org/action/displayAbstract?aid=46711.

31 Fan Wang. Cycle basis algorithms public repository. https://github.com/FanWang000/Cycle_basis_algorithms_public, 2026. Implementation and experimental scripts.

32 Dominic J Williamson and Theodore J Yoder. Low-overhead fault-tolerant quantum computation by gauging logical operators. *arXiv preprint*, 2024. arXiv:2410.02213.

33 Nicholas C Wormald et al. Models of random regular graphs. *London mathematical society lecture note series*, pages 239–298, 1999.

A The Frequency of Each Case in Version 3

We ran Version 3 on 25 random 3-regular graphs of size 8192 and 25 random 8-regular graphs of size 4096, and computed the average percentage of each case across the different graphs. The results show that Case 1 and Case 2B occur relatively infrequently compared to Case 2A and Case 3. As expected, when the graphs become denser, Case 3 becomes dominant.

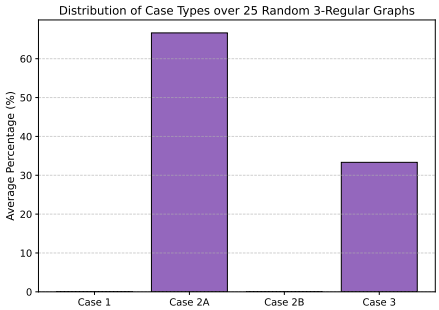

Figure 7 Average percentage of each case for 25 random 3-regular graphs of size 8192.

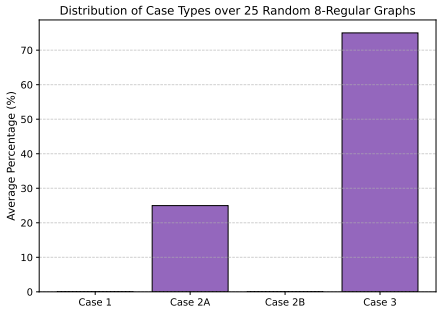

Figure 8 Average percentage of each case for 25 random 8-regular graphs of size 4096.

B Experimental Results for Graphs from Quantum Codes

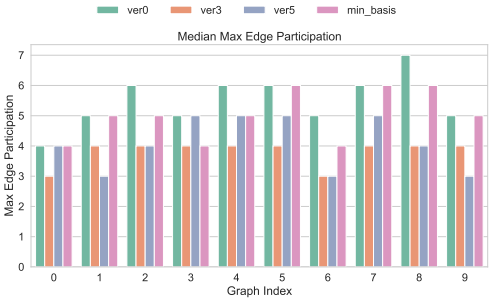

Figure 9 Median maximum edge participation across algorithmic variants for the graphs arising from the $[[250, 10, 15]]$ code.

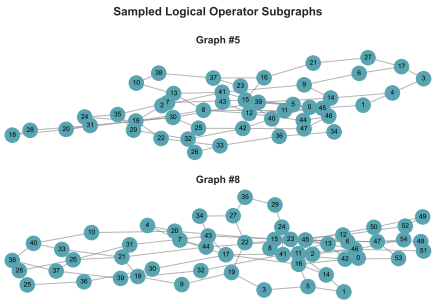

Figure 10 Two example logical-operator–induced subgraphs from the dataset. Each edge represents a qubit in the ancilla system used for lattice surgery, and each vertex corresponds to an X-check. The choice of cycle basis determines the associated set of Z-checks.

C Proofs

Proof of Lemma 2. For a single bucket, the multiplicative Chernoff lower tail bound

$$\Pr[X \le (1-\delta)\mu] \le \exp\left(-\frac{\delta^2\mu}{2}\right)$$

with $\delta = 1 - c$ gives: $p_j := \Pr[X_{j,i} \le c\mu_j] \le \exp\left(-\frac{(1-c)^2}{2}\mu_j\right)$. Since $\mu_j = k_j/6 \ge \log_2(\frac{2}{3}m_j)/3 = \ln(\frac{2}{3}m_j)/(3\ln 2)$, we have: $p_j \le \left(\frac{2}{3}m_j\right)^{-\alpha}$.

Let Y_i be the indicator that bucket i is bad. Then $B_j = \sum_i Y_i$ is a sum of Bernoulli variables with $\mathbb{E}[Y_i] \le p_j$. Hence, $\mathbb{E}[B_j] \le m_j \cdot p_j = \left(\frac{3}{2}\right)^{\alpha} m_j^{1-\alpha}$. Now apply Markov's inequality: $\Pr\left(B_j \ge \left(\frac{3}{2}\right)^{\alpha} m_j^{1-\frac{\alpha}{2}}\right) \le \frac{\mathbb{E}[B_j]}{\left(\frac{3}{2}\right)^{\alpha} m_j^{1-\frac{\alpha}{2}}} \le m_j^{-\frac{\alpha}{2}}$. ◀

Proof of Lemma 3. We first bound the total failure probability over all L epochs. Since $L = \lceil \frac{c}{2}\log_2 M \rceil \le \frac{c}{2}\log_2 M + 1$, we have $m_L = M/2^L \ge \frac{1}{2}M^{1-\frac{c}{2}}$. From Lemma 2, the probability that epoch j has more than $(\frac{3}{2})^{\alpha} m_j^{1-\frac{\alpha}{2}}$ bad buckets is at most $m_j^{-\alpha/2}$. Thus, the union bound gives:

$$\sum_{j=0}^{L-1} \Pr[\text{epoch } j \text{ fails}] \le \sum_{j=0}^{L-1} m_j^{-\alpha/2} \le L \cdot m_{L-1}^{-\alpha/2}.$$

Since $m_{L-1} \ge M^{1-\frac{c}{2}}$, we have

$$L \cdot m_{L-1}^{-\alpha/2} \le \left(\tfrac{c}{2}\log_2 M + 1\right) \cdot M^{-\beta}, \text{where } \beta := \tfrac{\alpha}{2}(1-\tfrac{c}{2}) > 0.$$

As $M \to \infty$, this product is $o(1)$, so with high probability all epochs satisfy the per-epoch bound.

Conditioned on this event, we now sum the bad buckets over all epochs:

$$\sum_{j=0}^{L-1} B_j \le (\tfrac{3}{2})^{\alpha} \sum_{j=0}^{L-1} m_j^{1-\frac{\alpha}{2}} = (\tfrac{3}{2})^{\alpha} M^{1-\frac{\alpha}{2}} \sum_{j=0}^{L-1} 2^{-j(1-\frac{\alpha}{2})}.$$

This geometric sum is bounded by

$$\sum_{j=0}^{\infty} 2^{-j(1-\frac{\alpha}{2})} = \frac{1}{1-2^{-(1-\frac{\alpha}{2})}}.$$

Hence the total number of bad buckets is at most $C(c)\, M^{1-\frac{\alpha}{2}}$, where $C(c) = (\frac{3}{2})^{\alpha} \cdot \frac{1}{1-2^{-(1-\frac{\alpha}{2})}}$. ◀

Proof of Theorem 1. Since each deleted bucket has a unique ancestor among the original M buckets, Lemma 3 bounds how many buckets *ever* go bad. Provided that $\alpha(c) > c$, which holds for every $c < 0.16$, we have $M^{1-\frac{\alpha}{2}} = o\left(M^{1-\frac{c}{2}}\right)$, and hence $C(c)\, M^{1-\frac{\alpha}{2}} < m_L$ for sufficiently large M, forcing the existence of at least one bucket that is never bad in any epoch $j < L$.

For such a never-bad bucket, its load in epoch j satisfies $X_j \ge c\,\mu_j \ge \frac{c}{3}\log_2\left(\frac{2}{3}m_j\right)$, using the lower bound $\mu_j \ge \frac{1}{3}\log_2\left(\frac{2}{3}m_j\right)$ from earlier.

Summing over $L = \left\lceil \frac{c}{2}\log_2 M \right\rceil$ epochs, we obtain a lower bound on the total load on that bucket:

$$\begin{aligned} L_{\max} &\ge \frac{c}{3}\sum_{j=0}^{L-1} \log_2\left(\tfrac{2}{3}m_j\right) = \frac{c}{3}\sum_{j=0}^{L-1}\left(\log_2 M - j + \log_2\tfrac{2}{3}\right) \\ &\ge \frac{c}{3}\cdot L \cdot \left(\log_2 M - L + 1 + \log_2\tfrac{2}{3}\right), \end{aligned}$$

where we use that $\log_2 m_j = \log_2(M\cdot 2^{-j}) = \log_2 M - j$ and that $\log_2\left(\frac{2}{3}m_j\right)$ is decreasing in j, so each term is at least the last one.

Substituting the bounds $\frac{c}{2}\log_2 M \le L \le \frac{c}{2}\log_2 M + 1$, we control both terms in the product:

$$\begin{aligned} L_{\max} &\ge \frac{c}{3}\cdot L \cdot \left(\log_2 M - L + 1 + \log_2\tfrac{2}{3}\right) \\ &\ge \frac{c}{3}\cdot\frac{c}{2}\log_2 M \cdot \left(\log_2 M - \left(\frac{c}{2}\log_2 M + 1\right) + 1\right) \\ &= \frac{c^2}{6}\left(1-\frac{c}{2}\right)\log_2^2 M. \end{aligned}$$

Thus, $L_{\max} = \Omega(\log^2 M)$, completing the proof of Theorem 1. ◀

D Process 1 Results in Higher Remaining Loads than Process 2

We define three new processes, P1, P1a, and P2, where P1 is essentially equivalent to Process 1 from the main text (as we are only interested in the loads of the remaining bins), and P2 is a slight variant of Process 2 in which only $k-3$ balls are thrown per round instead of k. The intermediate process P1a serves as a bridge to enable a direct, step-by-step comparison between P1 and P2.

▶ **Theorem 4** (Pointwise comparison of P1, P1a, and P2). *Let $m, k \in \mathbb{N}$ with $6 \mid m$, and let $L^{(0)} \in \mathbb{N}^m$ be any initial load vector. Run $T := m/6$ rounds of the following processes:*

P1: *At the start of each round, delete the three heaviest bins, then throw $k-3$ balls uniformly into the remaining bins.*

P1a: *In each round, mark the three heaviest bins, then throw $k-3$ balls uniformly into all bins. After round T, delete all marked bins.*

P2: *In each round, throw $k-3$ balls uniformly into all bins. After round T, delete the $m/2$ heaviest bins.*

For a random outcome R of the bin-selection randomness (i.e., the sequence of random bin selections in all rounds), let $L_{\mathrm{P}*}(R) \in \mathbb{N}^{m/2}$ denote the vector of surviving bin loads after deletion, sorted in non-increasing order. Then, for every R,

$$L_{\mathrm{P1}}(R) \geq L_{\mathrm{P1a}}(R) \geq L_{\mathrm{P2}}(R) \quad \textit{(entrywise).}$$

Proof. We couple all three processes (P1, P1a, and P2) using the same randomness. For each round $t = 1, \ldots, m/6$, we sample $k-3$ bins uniformly at random *without replacement* from $[m] = \{1, \ldots, m\}$, and record the selected labels as $S_t = (b_{t,1}, \ldots, b_{t,k-3})$. These selections are fixed and reused across all processes.

Comparison of P1a and P2. Since no bins are removed during the rounds of P1a or P2, both processes place balls identically in every round using the same selections S_t. Thus, before the final deletion step, P1a and P2 produce the same load vector. Process P2 deletes the $m/2$ bins with largest loads, which deterministically minimizes the remaining sorted load vector among all choices of $m/2$ bins. Therefore,

$$L_{\mathrm{P1a}}(R) \geq L_{\mathrm{P2}}(R) \quad \text{entrywise.}$$

Comparison of P1 and P1a. In P1, the three heaviest bins are deleted at the start of each round. Let $m_t = m - 3t$ be the number of remaining bins in round t. After this deletion, we relabel the remaining bins by sorting them in non-increasing order of their current loads and indexing them as $\{1, \ldots, m_t\}$.

For each label $b_{t,j} \in S_t$:

- If $b_{t,j} \leq m_t$, then the corresponding bin exists in both P1 and P1a during round t, and the ball is placed into the same bin in both processes.
- If $b_{t,j} > m_t$, then under the current relabeling this label corresponds to a bin that has already been deleted in P1 by round t. In P1, the ball is redirected to a bin chosen uniformly from $\{1, \ldots, m_t\}$, while in P1a the ball is placed into the original bin.

Thus, whenever a redirection occurs in P1, a ball that would have gone to a bin deleted in that round (under P1a) is instead placed into a bin that survives the round. Consequently, the load of every bin surviving P1 is at least as large as the load of the corresponding surviving bin in P1a. Since both processes delete exactly $m/2$ bins in total, after sorting we obtain

$$L_{\mathrm{P1}}(R) \geq L_{\mathrm{P1a}}(R) \quad \text{entrywise.}$$

Conclusion. Combining the two comparisons yields

$$L_{\mathrm{P1}}(R) \geq L_{\mathrm{P1a}}(R) \geq L_{\mathrm{P2}}(R),$$

as claimed. ◀

E BFS Results in Cycles with Logarithmic Length

We consider generating a random d-regular graph using the configuration model [33]. Consider the multi-set consisting of d copies of each $i \in [n]$. Edges are created by pairing items $2j-1$ and $2j$ in the ordering for $j = 1, \ldots dn/2$. If there are any multi-edges or self-loops, then the graph is discarded and the process repeats. $F(d,n)$ is defined to be the probability that the process succeed in producing a simple graph. It was proven in [1], that as n gets large, $F(d,n)$ converges to a constant that depends only on d and is independent of n.

In the proof below, we imagine generating the permutation and revealing the edges in the graph by exploring a BFS starting at vertex 1 until the first time a back edge, self-loop, or multi-edge is found. If the graph discovered so far is a simple tree, then BFS continues from some vertex v, be searching for the $d-1$ undiscovered occurrences of v in the permutation and examining the neighbors of v. The unrevealed items in the permutation are referred to as *unmatched half-edges*. If a back-edge is found before a self-loop or multi-edge, then the the rest of the graph is revealed to determine if the resulting graph is simple.

▶ **Theorem 5** (Logarithmic cycle from BFS with constant probability). *Fix $d \geq 3$. Consider the process described above to generate a d-regular random graph G. Then there exists a constant $c = c(d) > 0$ such that, for all sufficiently large n, with probability at least c the process produces a simple graph and the first back edge encountered creates a cycle of length $\Omega(\log n)$.*

Proof. We say that the BFS *aborts* when a back edge, self-loop, or multi-edge is found. We imaging running BFS in a series of iterations until it aborts. Before the first iteration, the BFS tree consists only of the single vertex 1. In each iteration, a leaf with the smallest depth is chosen, and all of its remaining $d-1$ neighbors are explored. Let A_t be the event that the BFS exploration aborts during or before iteration t, and define the conditional abort probability

$$P(t) := \Pr\left(A_t \,\middle|\, \bigcap_{i=1}^{t-1} A_i^c\right).$$

After $t-1$ successful iterations, the BFS tree contains

$$s = 1 + (t-1)(d-1)$$

vertices and $s-1$ exposed edges. Let S be the set of remaining half-edges. Then:

$$|S| = dn - 2(s-1).$$

Fix $T := n^{1/3}$. For all $t \leq T$, we have $s = O(n^{1/3})$ and thus $|S| = \Theta(n)$.

Condition on the explored BFS tree after $t-1$ iterations. The remaining edges are distributed as a uniformly random perfect matching on S. Let v be the leaf explored at iteration t.

Back edges / self-loops. For a fixed remaining half-edge of v, the probability that it is paired to a half-edge incident to one of the s currently discovered vertices is at most $O(s/|S|) = O(s/n)$. By a union bound over the $d-1$ half-edges of v, the probability of creating a back edge or self-loop is $O(ds/n)$.

Multi-edges. A multi-edge incident to v occurs if two of the $d-1$ half-edges of v are paired to half-edges of the same vertex. There are $\binom{d-1}{2} = O(d^2)$ such pairs. Fix one pair. Conditioned on the partner chosen for the first half-edge, the second hits the same vertex with probability at most $O(d/|S|) = O(d/n)$ (uniformly for $t \le T$ since $|S| = \Theta(n)$). Thus the probability of a multi-edge incident to v is $O(d^3/n)$.

Combining the failure modes,

$$P(t) \;\le\; O\!\left(\frac{ds}{n} + \frac{d^3}{n}\right) \;=\; O\!\left(\frac{d(s+d^2)}{n}\right).$$

Since $P(t)$ is defined conditionally,

$$\Pr\!\left(\bigcup_{t=1}^{T} A_t\right) = \sum_{t=1}^{T} P(t) \prod_{i=1}^{t-1} (1 - P(i)) \;\le\; \sum_{t=1}^{T} P(t).$$

Summing the bound using $s = \Theta(t)$ gives

$$\sum_{t=1}^{T} P(t) \;\le\; O\!\left(\frac{dT^2 + d^3T}{n}\right) = O(n^{-1/3}) = o(1).$$

Therefore the exploration does not abort in the first T iterations with probability $1 - o(1)$, and hence the probability of the event

$$\{\text{simple graph}\} \cap \{\text{no abort in first } T \text{ steps}\}$$

is at least $F(d,n) - o(1)$, which is bounded below by a positive constant depending only on d.

Conditioned on the BFS exploration remaining tree-like up to time T, the first back edge is incident to a uniformly random leaf of the BFS tree. Since the tree has $\Theta((d-1)^h)$ leaves at depth $h = \Theta(\log T)$, spread across the d root-subtrees, with constant probability the back edge connects two leaves whose least common ancestor is the root, thereby creating a cycle of length $\Omega(\log T) = \Omega(\log n)$. Combining with the constant lower bound on reaching T without abort proves the theorem. ◀