TheoretiCS

Brooks' Theorem in Graph Streams: A Single-Pass Semi-Streaming Algorithm for Δ-Coloring

Received Sep 11, 2022
Accepted Apr 13, 2023
Published August 3, 2023

Key words and phrases
Semi-streaming algorithms, Graph coloring, Brooks' theorem

Sepehr Assadi [a,b] ✉
Pankaj Kumar [c] ✉
Parth Mittal [a] ✉

a Department of Computer Science, Rutgers University, USA

b Cheriton School of Computer Science, University of Waterloo, Canada

c Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

ABSTRACT. Every graph with maximum degree Δ can be colored with $(\Delta+1)$ colors using a simple greedy algorithm. Remarkably, recent work has shown that one can find such a coloring even in the semi-streaming model: there exists a randomized algorithm that with high probability finds a $(\Delta+1)$-coloring of the input graph in only $O(n \cdot \text{polylog}\, n)$ *space* assuming a single pass over the edges of the graph in any arbitrary order. But, in reality, one almost never needs $(\Delta+1)$ colors to properly color a graph. Indeed, the celebrated **Brooks' theorem** states that every (connected) graph beside cliques and odd cycles can be colored with Δ colors. Can we find a Δ-coloring in the semi-streaming model as well?

We settle this key question in the affirmative by designing a randomized semi-streaming algorithm that given any graph, with high probability, either correctly declares that the graph is not Δ-colorable or outputs a Δ-coloring of the graph.

The proof of this result starts with a detour. We first (provably) identify the extent to which the previous approaches for streaming coloring fail for Δ-coloring: for instance, all these prior approaches can handle streams with repeated edges and they can run in $o(n^2)$ time, whereas

An extended abstract of this paper appeared in ACM Symposium on Theory of Computing (STOC'22) [**8**].
Sepehr Assadi was supported in part by an NSF CAREER Grant CCF-2047061, a Sloan Research Fellowship, a Google Research gift, and a Fulcrum award from Rutgers Research Council. Parth Mittal was supported in part by the NSF CAREER grant CCF-2047061. Part of this research was carried out while Pankaj Kumar and Parth Mittal participated in the 2020 DIMACS REU program, supported by CoSP, a project funded by European Union's Horizon 2020 research and innovation program, grant agreement No. 823748.

prove that neither of these tasks is possible for Δ-coloring. These impossibility results however pinpoint exactly what is missing from prior approaches when it comes to Δ-coloring.

We build on these insights to design a semi-streaming algorithm that uses (i) a novel sparse-recovery approach based on sparse-dense decompositions to (partially) recover the "problematic" subgraphs of the input—the ones that form the basis of our impossibility results—and (ii) a new coloring approach for these subgraphs that allows for recoloring of other vertices in a controlled way without relying on local explorations or finding "augmenting paths" that are generally impossible for semi-streaming algorithms. We believe both these techniques can be of independent interest.

1. Introduction

Graph coloring problems are ubiquitous in graph theory and computer science. Given a graph $G = (V, E)$, a proper c-coloring of G is any assignment of colors from the palette $\{1, \ldots, c\}$ to the vertices so that no edge receives the same color on both its endpoints. Recent years have witnessed a flurry of results for graph coloring in the graph streaming model [58, 15, 7, 13, 14, 4, 16, 6, 22, 39]. In this model, the edges of the input graph arrive one by one in an arbitrarily ordered stream and the algorithm needs to process these edges sequentially using a limited space, much smaller than the input size. Of particular interest are *semi-streaming* algorithms, introduced by [32], that use only $\widetilde{O}(n) := O(n \cdot \text{polylog}\, n)$ space[1] on n-vertex graphs which is proportional to the output size. We focus on this model in this paper.

One of the simplest forms of graph coloring problems is $(\Delta + 1)$-coloring of graphs with maximum degree Δ. Not only does every graph admits a $(\Delta + 1)$-coloring, one can in fact find one quite easily via a greedy algorithm: iterate over the vertices and color each one from any of $(\Delta + 1)$ colors that has not appeared in any of its at most Δ colored neighbors. Yet, despite its utter simplicity, this algorithm does not easily lend itself to a semi-streaming algorithm as the arbitrary arrival of edges prohibits us from coloring vertices one at a time.

Nonetheless, a breakthrough of [7] showed that $(\Delta + 1)$ coloring is still possible in the semi-streaming model, albeit via a randomized algorithm that employs a "non-greedy" approach. In particular, [7] proved the following *palette sparsification theorem*: if we sample $O(\log n)$ colors from $\{1, \ldots, \Delta + 1\}$ for each vertex independently, then with high probability, the entire graph can be colored by coloring each vertex from its own sampled colors. This result immediately leads to a semi-streaming algorithm for $(\Delta+1)$-coloring: sample these colors for each vertex and store any edge in the stream that can potentially become monochromatic under any coloring of vertices from their sampled list. A simple probabilistic analysis bounds the number of stored

1 Throughout, we use $\widetilde{O}(f) := O(f \cdot \text{poly}\log f)$ to suppress $\text{polylog}\,(f)$ factors.

edges by $O(n \log^2 n)$ with high probability, and the palette sparsification theorem guarantees that one can find a $(\Delta+1)$-coloring of the graph at the end of the stream.

Going back to existential results, it is easy to see that there are graphs that do need $\Delta+1$ colors for proper coloring, for instance $(\Delta+1)$-cliques or odd cycles (where $\Delta = 2$). The celebrated **Brooks' theorem** [19] states that these two are the only examples: any (connected) graph besides cliques and odd cycles can be colored with Δ colors (see also [49] and [47] for other classical proofs of this result by Melnikov and Vizing, and by Lovász, respectively). Unlike existence of $(\Delta+1)$-colorings which is rather a triviality, Brooks' theorem turned out to be a fundamental result in graph coloring [50, 63] with numerous proofs discovered for it over the years; see, e.g., [63, 25, 56, 57] and references therein. The algorithmic aspects of Brooks' theorem have also been studied extensively in classical algorithms [47, 62, 10], PRAM algorithms [44, 54, 43, 35], or LOCAL algorithms [53, 18, 34].

Given the key role Brooks' theorem plays in graph coloring literature on one hand, and the recent advances on streaming coloring algorithms on the other hand, it is thus quite natural to ask:

> *Does there exist a "**semi-streaming Brooks' theorem**", namely, a semi-streaming algorithm that colors any given graph, besides cliques and odd cycles, with Δ colors?*

This is precisely the question addressed in this paper. We emphasize that our interest in this question is not in "shaving off" a single color from $(\Delta+1)$-coloring to Δ-coloring in practice, but rather as a source of insights and ideas (as is the case, say, in graph theory or classical algorithms where $(\Delta+1)$-coloring is just a triviality). In fact, Δ-coloring appears to be just beyond the reach of our current techniques. For instance, previous streaming coloring algorithms in [7, 14, 4] can all be obtained via palette sparsification (see [4] for details). Yet, it was already observed in [7] that palette sparsification cannot handle Δ-coloring (we elaborate on this later). More generally, while $(\Delta+1)$-coloring has a strong "greedy nature", all existential/algorithmic proofs of Δ-coloring are based on "exploring" the graph for certain structures, say cut vertices or certain spanning trees [47], Kempe Chains [49], Rubin's Block Lemma [30, 5], or "augmenting paths" [54] to name a few (we refer the interested reader to [63] for an excellent overview of various proofs of Brooks' theorem). These (local) exploration tasks however tend to be generally impossible in the semi-streaming model.[2]

1.1 Our Contributions

We start with studying the *limitations* of the current approaches in streaming graph coloring for solving Δ-coloring. To do so, we focus on two common characteristics of all prior algorithms

2 For instance, while computing all neighbors of a given vertex is trivial via a semi-streaming algorithm (by storing edges of the vertex), it is even impossible to discover the 2-hop neighborhood of a given vertex [31].

in [7, 14, 4]: they all also naturally lead to (i) *sublinear-time* algorithms for the corresponding coloring problems that run in $(n^{3/2+o(1)})$ time, and (ii) semi-streaming algorithms that can handle *repeated-edge* streams wherein the same edge may appear more than once. We prove that obtaining either type of algorithms is provably *impossible* for Δ-coloring:

- **Sublinear-time algorithms (Appendix A.1):** Any algorithm that, given access to adjacency lists and adjacency matrix of a graph with known maximum degree Δ, can output a Δ-coloring with large constant probability requires $\Omega(n\Delta)$ queries to input and time.
- **Repeated-edge streams (Appendix A.2):** Any algorithm that, given the edges of a graph with known maximum degree Δ in a *repeated-edge* stream, can output a Δ-coloring with large constant probability requires $\Omega(n\Delta)$ space.

These impossibility results already demonstrate how different Δ-coloring is compared to prior graph coloring problems studied in the semi-streaming model. But, as we shall elaborate later, these results play a much more important role for us: they pinpoint what is missing from prior approaches when it comes to the Δ-coloring problem and act as an excellent guide for addressing our motivating question. This brings us to the main contribution of our work.

THEOREM 1.1 (Semi-Streaming Brooks' Theorem). *There exists a randomized semi-streaming algorithm that given any connected graph $G = (V, E)$ with maximum degree Δ, which is not a clique nor an odd-cycle, with high probability, outputs a Δ-coloring of G.*

Consequently, despite the fact that prior approaches inherently fail for Δ-coloring in fundamental ways and that Δ-coloring is provably intractable in closely related models, we can still obtain a semi-streaming Brooks' theorem and settle our motivating question in the affirmative. It is also worth mentioning that randomness in Theorem 1.1 is crucial: a very recent result of [6] shows that deterministic semi-streaming algorithms cannot even find an $\exp\left(\Delta^{o(1)}\right)$-coloring. Our Theorem 1.1 thus fully settles the complexity of the Δ-coloring problem in the semi-streaming model.

Theorem 1.1 can be stated more generally as an algorithm that either decides whether the input graph is Δ-colorable or not, and if yes, outputs the coloring. This is because checking whether a graph is Δ-colorable can be done by simply storing a spanning forest of the input (see, e.g., [32]) and maintaining the degrees of vertices; this allows us to check whether any of the connected components in the graph is a $(\Delta+1)$-clique or an odd-cycle. If not, applying Theorem 1.1 to each connected component of the graph (in parallel in a single pass) gives us the desired Δ-coloring (the algorithm does not even require the prior knowledge of Δ using a standard trick observed in [7]; see Remark 4.21). However, we find the statement of Theorem 1.1 to best capture the most interesting part of the result and thus opted to present it in this form.

Our Techniques. We shall go over our techniques in detail in the streamlined overview of our approach in Section 2. For now, we only mention the three main technical ingredients of our work:

i). A thorough understanding of the powers and limitations of the palette sparsification approach of [7] for Δ-coloring via a rough characterization of which (sub)graphs it still applies to;

ii). An algorithm for *implicitly* identifying and storing "problematic" subgraphs of the input graph—the ones that cannot be handled by palette sparsification approach of previous step—via a novel sparse recovery approach that relies on *algorithmic* use of sparse-dense decompositions (see Section 3.1) in place of their *analytic* use in prior streaming algorithms [7, 4];

iii). A new coloring procedure that combines simple graph theoretic ideas with probabilistic analysis of palette sparsification using a notion of *helper structures*; these are simple subgraphs of the input that can be recovered via our semi-streaming algorithms from the previous part and does not rely on local exploration steps of prior proofs of Brooks' theorem mentioned earlier.

Other Sublinear Algorithms Models. Prior semi-streaming algorithms for graph coloring also naturally lead to a series of algorithmic results for the respective problems in other models. Our first impossibility result already rules out this possibility for Δ-coloring when it comes to *sublinear-time* algorithms. Nevertheless, our approach in Theorem 1.1 is still quite flexible and thus allows for extension of this algorithm to many other models. In particular, the algorithm is implemented via a *linear sketch* (see [48]), which immediately implies the following two results as well:

— **Dynamic streams:** There exists a (single-pass) randomized semi-streaming algorithm for Δ-coloring on the streams that contain insertion and deletion of edges.

— **Massively parallel computation (MPC):** There exist a one round randomized MPC algorithm for Δ-coloring on machines of memory $\widetilde{O}(n)$ with only $\widetilde{O}(n)$ extra global memory.

As this is not the focus of the paper, we omit the definition and details of the models and instead refer the interested to [3, 48] and [45, 12] for each model, respectively.

1.2 Related Work

Similar to the classical setting, it is known that approximating the minimum number of colors for proper coloring, namely, the *chromatic number*, is intractable in the semi-streaming model [40, 2, 24]. Thus, recent work has focused instead on "combinatorially optimal" bounds—termed by [37]—for streaming coloring problems. On this front, we already discussed the $(\Delta+1)$-coloring

result of [7]. Independently and concurrently, [15] obtained a semi-streaming algorithm for $O(\Delta)$ colorings. These results were followed by semi-streaming algorithms for other coloring problems such as degeneracy coloring [14], coloring locally sparse graphs and $(\deg+1)$-coloring [4], $(\deg+1)$-list coloring [39], adversarially robust coloring [22], edge-coloring (in W-streams) [13], deterministic lower bounds and (multi-pass) algorithms [6], and coloring verification problems [16]. Moreover, [4] studied various aspects of palette sparsification technique of [7] and showed that other semi-streaming coloring algorithms such as [15, 14] can also be obtained via this technique.

Many of these work on streaming algorithms for graph coloring also extend to other models such as sublinear-time and massively parallel computation (MPC) algorithms. For instance, for $(\Delta+1)$-coloring, there are randomized sublinear-time algorithms in $\widetilde{O}(n^{3/2})$ time [7] or deterministic MPC algorithms with $O(1)$ rounds and $O(n)$ per-machine memory [27] (see also [26]). Moreover, subsequent to the conference publication of this paper in [8], some of the ideas in our work was also used in [33] in designing distributed LOCAL algorithms for Δ-coloring.

Numerous beautiful algorithmic results are known for Δ-coloring problem in various other models such as classical algorithms [47, 62, 10], PRAM algorithms [44, 54, 43, 35], or LOCAL algorithms [53, 18, 34, 11]. For instance, a remarkable "distributed Brooks' theorem" of [54] proves that any partial Δ-coloring of all but one vertex of the graph, can be turned into a proper Δ-coloring of the entire graph by recoloring a single "augmenting path" of $O(\log_\Delta n)$ length. Finally, it is worth mentioning that Brooks' theorem is part of a more general phenomenon in graph theory: as the maximum clique size in G moves further away from $\Delta+1$, so does its chromatic number. For instance, [60] proves that for sufficiently large Δ, if a graph does not contain a Δ-clique, then it is in fact always $(\Delta-1)$-colorable; see, e.g., [17, 59, 60, 46, 50, 52] and references therein for various other examples.

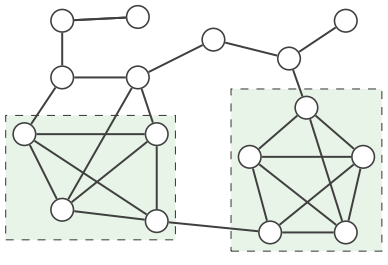
2. Technical Overview

We now give a streamlined overview of our approach. While Theorem 1.1 is by far the main contribution of our work, we find it illuminating to first talk about our impossibility results for Δ-coloring as they, despite their simplicity, played a crucial role for us in obtaining Theorem 1.1 and we believe they can shed more light into different components of our final algorithm.

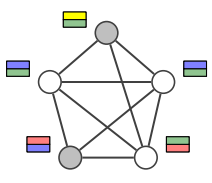
2.1 A Detour: Impossibility Results, Barriers, and Lessons Along the Way

Palette sparsification. Let us start by reviewing the palette sparsification theorem of [7]: if we sample $O(\log n)$ colors from $\{1,\ldots,\Delta+1\}$ for each vertex independently, then with high probability, we can still color the graph by coloring each vertex from its sampled palette. The proof of this result in [7] uses a variant of *sparse-dense decomposition* [59] that partitions the

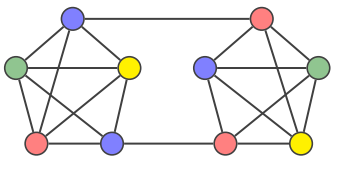
graph into *"(locally) sparse"* vertices and a collection of *"almost-clique"* subgraphs that can be turned into $(\Delta+1)$-cliques by changing $o(1)$ fraction of their vertices and edges (Figure 1a). The sparse vertices are then colored *one at a time* from their sampled lists using a standard greedy coloring argument originally introduced in [51]. The main part of the proof is to handle almost-cliques by going over them one by one and coloring each one *entirely*, using the sampled lists of *all* its vertices at the same time, even assuming the outside vertices are colored *adversarially*.



(a) Sparse-dense decomposition



(b) palette sparsification of Δ-coloring



(c) query lower bound of Δ-coloring

Figure 1. A graph with maximum degree $\Delta = 4$ and its sparse-dense decomposition in **(a)** (each box denotes an almost-clique and remaining vertices are sparse). Part **(b)** is an illustration of why palette sparsification fails for Δ-coloring: the only way to Δ-color this graph is to color the marked vertices the same, which cannot be done with these sampled lists. Part **(c)** shows a similar construction can be used to prove a query lower bound for Δ-coloring. (The actual instance is obtained from $\Theta(n/\Delta)$ copies of such pairs.)

As expected, the hard part in extending palette sparsification theorem of [7] to Δ-coloring involves the argument for almost-cliques. Indeed, this is not just a matter of analysis; as already observed by [7], this theorem fails for Δ-coloring (Figure 1b): consider a $(\Delta+1)$-clique minus a single edge (u, v); the only way we can find a Δ-coloring of this graph is if we color both u and v the same, which requires their sampled lists to intersect; by the birthday paradox this only happens when size of each list is $\Omega(\sqrt{\Delta})$ which in turn implies that the algorithm has to store $\Omega(n\Delta)$ edges from the stream—this is effectively the same as storing the input itself!

Sublinear time (query) algorithms. Consider a graph G which is a collection of $\Theta(n/\Delta)$ *pairs* of $(\Delta+1)$-cliques. For each pair, randomly pick two vertices (u_1, v_1) and (u_2, v_2) from its first and second clique, respectively. Remove the edges (u_1, v_1) and (u_2, v_2) and instead include the edges (u_1, v_2) and (u_2, v_1) in the graph. See Figure 1c for an illustration. It is easy to see that the only way to Δ-color this graph is to find the "switched" edges in each copy and color their endpoints the same inside each (now) almost-clique. Yet, it is an easy exercise to use the linear lower bound on the query complexity of OR function [20] to prove that this requires making $\Omega(\Delta^2)$ queries to the adjacency lists or matrix of the graph for each pair, and thus $\Omega(n\Delta)$ queries overall. This lower bound now leaves us with the following lesson.

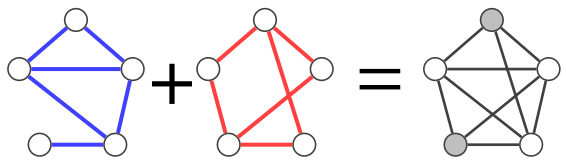


Figure 2. An illustration of the hard instances for the repeated-edge stream lower bound—the actual instance is obtained from $\Theta(n/\Delta)$ copies of these graphs. The only possible Δ-coloring is to color both endpoints of marked vertices the same.

LESSON 2.1. *Any semi-streaming algorithm for Δ-coloring should explicitly* look at *all but a tiny fraction of edges of the graph presented in the stream.*

Lesson 2.1 may sound trivial at first. After all, the semi-streaming model allows all algorithms to look at all edges of the graph. Yet, note that numerous semi-streaming algorithms, say, sampling algorithms, including all prior streaming coloring algorithms in [7, 14, 4, 39], do not use this power—Lesson 2.1 implies that these algorithms cannot solve Δ-coloring.

Semi-streaming algorithms on repeated-edge streams. What if we take Lesson 2.1 to the extreme and give the algorithm each edge (potentially) multiple times in the stream—this should surely helps us even more, no? It turns out however that this is not really the case.

Suppose that we have a graph G on a collection of $\Theta(n/\Delta)$ disjoint sets of vertices of size $\Delta+1$ each. For each set of $\Delta+1$ vertices U, consider a stream of edges that in the first part, provides a subset E_1 of edges over U and in the second part, provides another subset E_2—the repeated-edge stream allows these subsets to be overlapping and we shall choose them so that $E_1 \cup E_2$ leaves precisely one pair of vertices (u,v) among all pairs in U without an edge. As before, the only way to Δ-color this graph is to color vertices u, v in each of the $\Theta(n/\Delta)$ pieces the same. But, given that the edges between E_1 and E_2 may overlap, one can prove that finding all these pairs requires $\Omega(n\Delta)$ space. This is by a reduction from communication complexity lower bounds of the *Tribes* function [42] (a slightly less well-known cousin of the famous set disjointness problem). This brings us to the next lesson.

LESSON 2.2. *Any semi-streaming algorithm for Δ-coloring should crucially use the fact that each edge of the graph arrives* exactly once *in the stream.*

Again, while the semi-streaming model only allows for presenting each edge once in the stream, many algorithms are entirely oblivious to this feature. This includes all previous semi-streaming coloring algorithms in [7, 15, 14, 4, 39], as well as various other ones for spanning trees [32], sparsifiers [48], spanners [32, 31] and maximal matchings [32]. Lesson 2.2 says that any potential Δ-coloring algorithm cannot be of this type.

A natural algorithm or a barrier result? Finally, let us conclude this part by considering a natural semi-streaming algorithm for Δ-coloring: Sample $O(n \log n/\Delta)$ vertices S uniformly at random, and partition the input graph into two subgraphs G_S consisting of all edges incident on S, and G_{-S} consisting of all remaining edges. We can easily detect, for each arriving edge in the stream, which subgraph it belongs to. Moreover, it is easy to see that G_S contains $O(n \log n)$ edges and G_{-S}, with high probability, has maximum degree at most $\Delta - 1$. We can thus store G_S explicitly via a semi-streaming algorithm and run the algorithm of [7] on G_{-S} to color it with $(\Delta(G_{-S}) + 1) = \Delta$ colors. So, we have a Δ-coloring of G_{-S} and all edges of G_S.

Surely, now that we know *all* of G_S, we should be able to extend the Δ-coloring of G_{-S} to G_S, no? The answer however turns out to be *no*: unlike the case of $(\Delta + 1)$-coloring, not every partial coloring of a graph can be extended directly to a proper Δ-coloring of the entire graph. But perhaps this is only an abstract worry and we should just find the right way of analyzing this algorithm? The answer is yet again *no*: the algorithm we just proposed in fact neglects both Lesson 2.1 and Lesson 2.2 and thus is doomed to fail completely.[3] But this also leaves us with the following lesson.

LESSON 2.3. *Any semi-streaming algorithm for Δ-coloring that colors the graph by extending a partial coloring, subgraph by subgraph, should either provide a* stronger guarantee *than solely an arbitrary Δ-coloring for each subgraph, or allow for* recoloring *of an already colored subgraph.*

While perhaps less concrete than our two previous lessons, Lesson 2.3 has a profound impact in the design of our semi-streaming algorithm that also colors the graph one subgraph at a time; in particular, our algorithm is going to adhere to *both* restrictions imposed by Lesson 2.3 simultaneously.

2.2 The High-Level Overview of Our Algorithm

After this long detour, we are now ready to go over our algorithm in Theorem 1.1. As stated earlier, the three main ingredients of our algorithm are: (i) a variant of palette sparsification for Δ-coloring that can color all but some problematic almost-cliques in the input (such as Figure 1b), (ii) a sparse recovery approach for (partially) recovering these problematic subgraphs, and (iii) a new coloring procedure that allows for extending the partial coloring of part (i) to the remaining subgraphs partially recovered in part (ii) to obtain a proper Δ-coloring of the entire graph. We will go over each part separately in the following.

Part One: Powers and Limitations of Palette Sparsification for Δ-Coloring

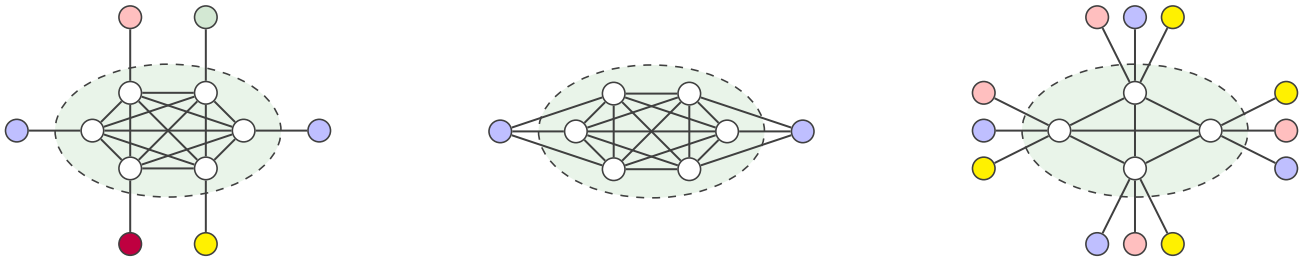
While we already discussed in Section 2.1 that palette sparsification fails for Δ-coloring, we are still going to employ its ideas crucially in our work. The goal of this step is to identify to

3 An added bonus of those impossibility results is to allow for quickly checking viability of potential algorithms.

what extent this approach fails for Δ-coloring. We develop a **classification of almost-cliques** (Section 4.1) based on the following three criteria:

— **Size**—number of vertices: *small* for $< \Delta + 1$, *critical* for $\Delta + 1$, and *large* for $> \Delta + 1$ vertices.
— **Inner density**—number of non-edges inside the almost-clique: *holey* for $\Omega(\Delta)$ non-edges (or "holes" in the almost-clique), and *unholey* otherwise.
— **Outer connections**—a measure of how "tightly" the almost-clique is connected to outside; we postpone the technical details of this part to the actual definition in Section 4.1.

Among these, size and inner density are perhaps usual suspects. For instance, we already saw in Section 2.1 that palette sparsification entirely fails for almost-cliques of size $\Delta + 1$ with exactly one non-edge—in our classification, these correspond to critical unholey almost-cliques. The third criterion is more technical and is motivated by Lesson 2.3; as we are still going to color the graph one almost-clique at a time, we would like to be able to reason about the partial coloring of outside vertices and possibility of its extension to the almost-clique. This is particularly relevant for small almost-cliques which can be actually a true clique inside and hence would definitely be in trouble if the same exact set of colors is "blocked" for all their vertices from outside. See Figure 3.



(a) "random" outside-coloring **(b)** "adversarial" outside-coloring **(c)** "adversarial" outside-coloring

Figure 3. An illustration of three possible types of outer connections on a graph with maximum degree $\Delta = 6$. The almost-clique in part **(a)** has a "right" type of outside connection and is going to receive a more "random" coloring on its neighbors, compared to the almost-clique in part **(b)** with "few" outside neighbors and part **(c)** with "too many" ones. In particular, the latter almost-cliques now cannot be Δ-colored without changing the color of outside vertices as the same colors are blocked for all vertices of the inner (actual) cliques.

We then consider palette sparsification (on steroids!) wherein each vertex samples $\text{polylog}(n)$ colors from $\{1, \ldots, \Delta\}$ and *characterize* which families of almost-cliques in our classification can still be colored using only the sampled colors. We show that all holey almost-cliques (regardless of their size or outer connections) can be still colored from their sampled colors using a similar argument as in [7]. More interestingly, we show that even unholey small almost-cliques that have the "right" type of outside connections can be colored at this step. Our analysis in this part deviates significantly from [7] and in particular crucially establishes

certain *randomness* properties on the coloring of vertices *outside* of an almost-clique when trying to color the almost-clique itself (recall Lesson 2.3). Thus, what remains are unholey critical almost-cliques (regardless of their outer connections) and unholey small almost-cliques with "problematic" outside connections. We delegate coloring of these almost-cliques to the next steps of the algorithm.

Let us now briefly discuss the effect of outer connections in coloring a small almost-clique. As stated earlier, the main problem with small almost-cliques occurs when exactly the same set of colors is used to color all outside vertices, thus blocking these colors entirely for the almost-clique. While this event is basically unavoidable for almost-cliques with only a couple of outside neighbors (Figure 3b), it becomes less and less likely as the number of outside neighbors increases (Figure 3a). After all, for a color to be used on all these vertices, it should be sampled by every single one of them in the first place. We will however run into problem again in cases when we have "too many" outside neighbors for every single vertex of the almost-clique (Figure 3c). The almost-cliques of Figure 3c are particularly problematic as we have no knowledge of the neighborhood of their outside vertices (for Figure 3b, we at least know that each of them have many neighbors in the almost-clique, which is used crucially by our latter algorithms). Thus, we should basically avoid ending up in a situation that we have to color such almost-cliques after having colored their outside neighbors.

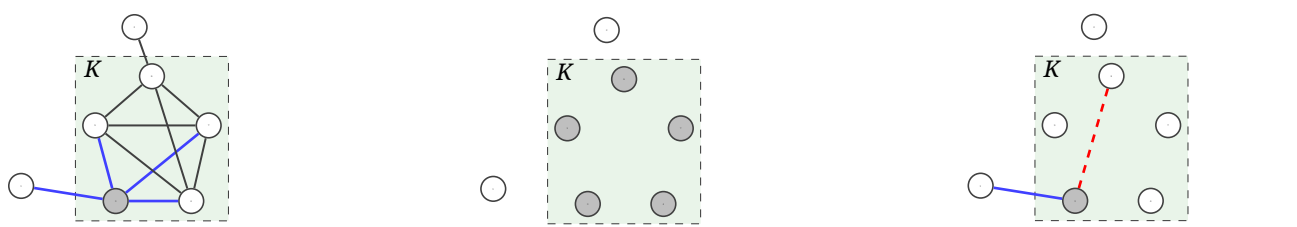
Fortunately, the almost-cliques of Figure 3c can only happen for small enough almost-cliques; this in turn makes the neighborhood of these almost-cliques sufficiently sparse. Thus, we can instead handle them similar to sparse vertices by *increasing* the size of sampled palettes for vertices. There is however a subtle issue with this approach. Increasing the size of sampled palettes means that even a fewer number of outside vertices can make a problem for us, hence requiring us to send even more almost-cliques to sparse vertices to handle, leading to a chicken-and-egg problem. A key idea in this part is a way to break this dependency cycle by careful sequencing the order of processing of vertices in a way that ensures sufficient "randomness" exist in the coloring of neighborhood of all small almost-cliques, except for the ones with very few outside neighbors (the type in Figure 3b). We postpone the discussion of this "dependency-breaking" step to Section 5 and Remark 5.7.

All in all, this step effectively establishes that palette sparsification achieves a "weak" streaming Brooks' theorem by Δ-coloring graphs that do not contain certain *forbidden* subgraphs such as $(\Delta+1)$-cliques minus few edges or Δ-cliques that have few neighbors outside (for Brooks' theorem itself, the only forbidden subgraph is a $(\Delta+1)$-clique).

Part Two: Sparse Recovery for Remaining Almost-Cliques, and Helper Structures

Our next step is a way of *finding edges of* the almost-cliques left uncolored by the previous step so that we can color them using a different approach. Let us bring up an obvious point here: these left out almost-cliques are precisely the same family of instances that were at the

core of our impossibility results in Section 2.1 (and their natural relaxations). Consequently, to handle them, our algorithm needs to take into account the recipe put forward by Lesson 2.1 and Lesson 2.2: it should look at *all* edges of the stream *exactly once*. This rather uniquely points us toward a canonical technique in the streaming model: **sparse recovery** (via linear sketching).



(a) neighbors-vector of $v = 3$ **(b)** characteristic-vector of K **(c)** recovery-vector for neighbors of 3

Figure 4. An illustration of sparse-recovery on the neighborhood of each vertex, plus an algorithm that finds the identity of vertices in each almost-clique, allows for recovering all edges "highly-dense" almost-cliques. Our actual algorithm is considerably more involved as it needs to *partially* recover "not-too-dense" almost-cliques also.

Consider an almost-clique which is a $(\Delta+1)$-clique minus an edge (Figure 1b). On the surface, recovering all edges of this almost-clique is problematic as these subgraphs are actually quite *dense*; for instance, if the graph consists of only copies of such almost-cliques, we will need $\Omega(n\Delta)$ space to store all of them. But what saves us at this stage is the fact that these subgraphs are actually too dense! Informally speaking, this reduces their "entropy" dramatically conditioned on our knowledge of the sparse-dense decomposition. Thus, we can recover them *implicitly* using a novel sparse recovery approach that uses sparse-dense decompositions algorithmically and not only analytically.[4]

Although in the examples we discussed, we can hope to recover the entire almost-clique in question implicitly, this will not be the case for all almost-cliques left uncolored by the first part, e.g., for a $(\Delta+1)$-clique minus a $\sqrt{\Delta}$-size inner clique (applying this method to a graph consisting of only such almost-cliques requires $\Omega(n\sqrt{\Delta})$ space). As a result, our semi-streaming algorithm settles for recovering certain **helper structures** from these almost-cliques instead. These are subgraphs of the input that are sufficiently simple to be recoverable via a combination of sparse recovery, sampling, and some basic graph theory arguments. At the same time, they are structured enough to give us enough flexibility for the final coloring step.

4 For the main results in [7] one only needs to know the *existence* of the decomposition and does not need to compute it. That being said, [7] also gave algorithms for finding the decomposition from the stream (which is needed to run their algorithms in polynomial time)—we use an extension of their algorithm by [9] in this paper.

Given the technicality of their definitions (Definitions 4.6 and 4.7 and Figure 7), we postpone further details to Section 4.1.

We point out that sparse recovery and linear sketching have been a staple of graph streaming algorithms since the seminal work of [3]. But, these tools have been almost exclusively used to handle edge deletions in *dynamic* streams. Their applications for us, on *insertion-only* streams, as a way of (implicitly) sparsifying a graph using outside information (i.e., the sparse-dense decomposition), is quite different and can form a tool of independent interest.

Part Three: The Final Coloring Procedure

The final step of our approach is then to color these remaining almost-cliques, given the extra information we recovered for them in the previous step. For intuition, let us consider two inherently different types of almost-cliques left uncolored by the approach in the first part.

(a) a $(\Delta+1)$-clique minus an edge **(b)** a Δ-clique with few neighbors

Figure 5. Two problematic almost-cliques in a graph with maximum degree $\Delta = 4$. Both almost-cliques are *hard* for palette sparsification. The only way part **(a)** can be Δ-colored is if the vertices incident on the non-edge have intersecting lists. Part **(b)** is *not* Δ-colorable if the outside (marked) vertices are all colored the same.

$(\Delta+1)$-clique minus an edge (u, v) (Figure 5a): Suppose we know all edges of such an almost-clique. We can color both u and v the same using a color that does not appear in their outside neighbors (which is possible because vertices of almost-cliques have few edges out); the standard greedy algorithm now actually manages to Δ-color this almost-clique (recall that we assumed the knowledge of *all* edges of the almost-clique for now). The argument is simply the following: Pick a common neighbor z of u and v, which will exist in an almost-clique, and greedily color vertices from a color not used in their neighborhood, waiting for z to be colored last; at this point, since two neighbors of z have the same color, there is still a choice for z to be colored with in the algorithm.

Δ-clique with few neighbors (Figure 5b): These are more problematic cases if we have ended up coloring all their outside neighbors the same. Even if we know all edges of this almost-clique, there is no way we can color a Δ-clique with $(\Delta-1)$ colors (the same one color is "blocked"

for all vertices). So, the next ingredient of our algorithm is a **recoloring step** that allows for handling these almost-cliques (again, recall Lesson 2.3); we show that our strengthened palette sparsification is flexible enough that, given the edges of a new almost-clique, allows for altering some of its *past* decisions on outside vertices of this almost-clique. This step involves reasoning about a probabilistic process (possibility of having a "good" color to change for an outside vertex) *after* already viewing the outcome of the process (having ended up with a "blocked" color). This requires a careful analysis which is handled by partitioning the randomness of the process into multiple *phases* and using our classification of almost-cliques to limit the amount of "fresh randomness" we need for this step across these phases. We discuss this in more details in Section 5.

The discussion above oversimplified many details. Most importantly, we actually do not have such a "clean" picture as above for the remaining almost-cliques we need to color. Instead, the algorithm needs to handle almost-cliques that are not fully recoverable by sparse recovery (as they are not sufficiently dense), using their helper structures described earlier.[5] This coloring is thus done via a combination of the greedy arguments of the above type on the helper structures, combined with palette sparsification ideas for the remaining vertices of the almost-clique. This in turns requires using some **out of (sampled) palette coloring** of vertices, which is in conflict with what the palette sparsification does and needs to be handled carefully; see Sections 5.5 and 5.6.

3. Preliminaries

Notation. For an integer $t \geqslant 1$, we define $[t] := \{1, 2, \ldots, t\}$. For a graph $G = (V, E)$ and a vertex $v \in V$, we use $N_G(v)$ to denote the neighbors of v in G, and $N(v)$ when the graph is clear from the context. Further, we define degree of v by $\deg_G(v) := |N_G(v)|$ (and similarly $\deg(v)$). We refer to a pair (u, v) of vertices in V as a **non-edge** when there is no edge between u and v in G. Similarly, we sometimes say that u is a **non-neighbor** of v and vice versa.

For a graph $G = (V, E)$ and integer $q \geqslant 1$, we refer to any function $C : V \to [q]$ as a q**-coloring** and call it a *proper* coloring iff there is no edge (u, v) in G with $C(u) = C(v)$. We further refer to a function $C : V \to [q] \cup \{\perp\}$ as a **partial** q**-coloring** and call the vertices $v \in V$ with $C(v) = \perp$ as **uncolored** vertices by C. The edges (u, v) in G with $C(u) = C(v) = \perp$ are *not* considered monochromatic, and thus we consider a partial q-coloring proper iff there is no edge (u, v) in G with $C(u) = C(v) \neq \perp$. Finally, for any proper partial q-coloring, and any vertex $v \in V$, we define:

— $\mathsf{ColNei}_C(v) := \{u \in N(v) \mid C(u) \neq \perp\}$: the neighbors of v that are assigned a color by C; we further define $\mathsf{coldeg}_C(v) := |\mathsf{ColNei}_C(v)|$.

5 For a $(\Delta+1)$-clique minus an edge (u, v), the helper structure is the subgraph consisting of vertices u and v, and all edges incident on at least one of them. The helper structure of the other example is more tricky; see Figure 7.

— $\mathsf{Avail}_C(v) := \{c \in [q] \mid C(u) \neq c \text{ for all } u \in N(v)\}$: the colors in $[q]$ that have not been assigned to any neighbor of v by C, i.e., are *available* to v; we define $\mathsf{avail}_C(v) := |\mathsf{Avail}_C(v)|$.

Finally, we say that a coloring C_1 is an **extension** of coloring C_2 iff for every $v \in V$ with $C_2(v) \neq \perp$, $C_1(v) = C_2(v)$; in other words, only uncolored vertices in C_2 may receive a different color in C_1.

For a bipartite graph $H = (L, R, E)$, a matching M is any collection of *vertex-disjoint* edges. We say that a matching M is an L-**perfect matching** iff it matches all vertices in L. We use the following presentation of the well-known Hall's marriage theorem [36].

FACT 3.1 (cf. [36]). *Suppose $H = (L, R, E)$ is a bipartite graph such that for any set $A \subseteq L$, we have $|N(A)| \geqslant |A|$; then H has an L-perfect matching.*

Concentration results. We use the following standard form of Chernoff bound in our proofs.

PROPOSITION 3.2 (Chernoff bound; c.f. [29]). *Suppose $X_1, \ldots, X_m$ are m independent random variables with range $[0, b]$ each. Let $X := \sum_{i=1}^{m} X_i$ and $\mu_L \leqslant \mathbb{E}[X] \leqslant \mu_H$. Then, for any $\delta > 0$,*

$$\Pr\left(X > (1+\delta)\cdot\mu_H\right) \leqslant \exp\left(-\frac{\delta^2\cdot\mu_H}{(3+\delta)\cdot b}\right) \quad \text{and} \quad \Pr\left(X < (1-\delta)\cdot\mu_L\right) \leqslant \exp\left(-\frac{\delta^2\cdot\mu_L}{(2+\delta)\cdot b}\right).$$

Throughout, we say that an event happens *"with high probability"*, or *"w.h.p."* for short, to mean that it happens with probability at least $1 - 1/\mathrm{poly}(n)$ for some large polynomial (the degree can be arbitrarily large without changing the asymptotic performance of the algorithms). Moreover, we pick this degree to be large enough to allow us to do a union bound over the polynomially many events considered and we do not explicitly mention this union bound each time (but in certain places that we need to do a union bound over exponentially many events, we will be more explicit).

3.1 A Sparse-Dense Decomposition

We use a simple corollary of known streaming sparse-dense decompositions in [7, 9], which have their origin in the classical work in graph theory [59, 50] and have subsequently been used extensively in distributed algorithms as well [41, 23, 55, 38].

The decomposition is based on partitioning the vertices into *"(locally) sparse"* vertices with many non-edges among their neighbors and *"almost-clique"* vertices that are part of a subgraph which is close to being a clique. We formally define these as follows.

DEFINITION 3.3. For a graph $G = (V, E)$ and parameter $\varepsilon > 0$, a vertex $v \in V$ is ε-**sparse** iff there are at least $\varepsilon^2 \cdot \Delta^2/2$ non-edges between the neighbors of v.

DEFINITION 3.4. For a graph $G = (V, E)$ and parameter $\varepsilon > 0$, a subset of vertices $K \subseteq V$ is an ε-**almost-clique** iff K has the following properties:

i). Size of K satisfies $(1-5\varepsilon)\cdot\Delta \leqslant |K| \leqslant (1+5\varepsilon)\cdot\Delta$.

ii). Every vertex $v \in K$ has $\leqslant 10\varepsilon\Delta$ *non-neighbors* inside K;

iii). Every vertex $v \in K$ has $\leqslant 10\varepsilon\Delta$ *neighbors* outside K;

iv). Every vertex $u \notin K$ has $\geqslant 10\varepsilon\Delta$ *non-neighbors* inside K.

We note that property *iv*). of Definition 3.4 does not typically appear in the definition of almost-cliques in prior work but it is crucial for our proofs. However, this property follows immediately from the proof of [9]. We have the following sparse-dense decomposition.

PROPOSITION 3.5 (Sparse-Dense Decomposition; cf. [50, 7, 9]). *There is a constant $\varepsilon_0 > 0$ such that the following holds. For any $0 < \varepsilon < \varepsilon_0$, vertices of any graph $G = (V, E)$ can be partitioned into* **sparse vertices** V_{sparse} *that are ε-sparse (Definition 3.3) and* **dense vertices** *partitioned into a collection of disjoint ε-almost-cliques $K_1, \ldots, K_k$ (Definition 3.4).*

Moreover, there is an absolute constant $\gamma > 0$ and an algorithm that given access to only the following information about G, with high probability, computes this decomposition of G:

- **Random edge samples:** *A collection of sets $N_{\text{sample}}(v)$ of $(\gamma \cdot \varepsilon^{-2} \cdot \log n)$ neighbors of every vertex $v \in V$ chosen independently and uniformly at random (with repetition);*
- **Random vertex samples:** *A set* SAMPLE *of vertices wherein each $v \in V$ is included independently with probability $(\gamma \cdot \log n/\Delta)$, together with all the neighborhood $N(v)$ of each sampled vertex $v \in$ SAMPLE.*

We note that a dense vertex is *not* necessarily not ε-sparse. Or in other words, the set V_{sparse} may not include all the ε-sparse vertices of G, as some ε-sparse vertices may still be be included in some ε-almost-clique.

3.2 Sparse Recovery

We also use the following standard variant of sparse recovery in our proofs. We note that the specific recovery matrix below, the Vandermonde matrix, is not necessary for our proofs (i.e., can be replaced with any other standard construction) and is only mentioned explicitly for concreteness.

PROPOSITION 3.6 (cf. [28]). *Let $n, k \geqslant 1$ be arbitrary integers and $p \geqslant n$ be a prime number. Consider the $(2k \times n)$-dimensional Vandermonde matrix over $\mathbb{F}_p$:*

$$\Phi^{\text{V}} := \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ 1 & 2^2 & 3^2 & \cdots & n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{2k-1} & 3^{2k-1} & \ldots & n^{2k-1} \end{bmatrix}, \quad \text{or for all } i \in [2k] \text{ and } j \in [n]: \quad \Phi^{\text{V}}_{i,j} = j^{i-1} \bmod p.$$

Then, for any k-sparse vector $x \in \mathbb{F}_p^n$, one can uniquely recover x from $\Phi^{\mathrm{V}} \cdot x$ in polynomial time.

The proof that a k-sparse vector x can be recovered from $\Phi^{\mathrm{V}} \cdot x$ is simply the following: for $\Phi^{\mathrm{V}} \cdot x = \Phi^{\mathrm{V}} \cdot y$ for two k-sparse vectors $x \neq y$ (i.e. recovery is impossible), we need $\Phi^{\mathrm{V}} \cdot (x - y) = 0$. But then this means that the (at most) $2k$ columns in the matrix Φ^{V} corresponding to the support of $x - y$ have a non-trivial kernel; the latter is a contradiction as any $2k$ columns of Φ^{V} are independent (polynomial time recovery also can be obtained via syndrome decoding from coding theory).

In order to safely use sparse recovery (in case when we mistakenly run it on a non-sparse vector), we also need the following standard result that allows us to test whether the output of the recovery is indeed correct or not. This result is also standard and is proven for completeness.

PROPOSITION 3.7. *Let $n, t \geqslant 1$ be arbitrary integers and $p \geqslant n$ be a prime number. Consider the $(t \times n)$-dimensional random matrix Φ^{R} over $\mathbb{F}_p$ chosen uniformly from all matrices in $\mathbb{F}_p^{t \times n}$. Then, for any two different vectors $x \neq y \in \mathbb{F}_p^n$, we have,*

$$\Pr_{\Phi^{\mathrm{R}}}\left(\Phi^{\mathrm{R}} \cdot x = \Phi^{\mathrm{R}} \cdot y\right) = p^{-t}.$$

PROOF. Let $z := x - y$ and note that there exists at least one index $i \in [n]$ with $z_i \neq 0$. We need to calculate the probability of $\Phi^{\mathrm{R}} \cdot z = 0$. Let r be any row of matrix Φ^{R}. We have,

$$\Pr\left(\langle r, z\rangle = 0\right) = \Pr\left(r_i \cdot z_i = \sum_{j \neq i} r_j \cdot z_j\right) = p^{-1},$$

because $r_i \cdot z_i$ is going to be any element of the field $\mathbb{F}_p$ with equal probability (as z_i is non-zero) and choice of r_i is independent of $\{r_j \mid j \neq i\}$. As all the t rows of Φ^{R} are independent, we get the final bound immediately. ■

We can now combine Proposition 3.6 and Proposition 3.7 to have a "safe" recovery w.h.p. as follows: For the (unknown) vector $x \in \mathbb{F}_p^n$, we compute $\Phi^{\mathrm{V}} \cdot x$ and $\Phi^{\mathrm{R}} \cdot x$ in parallel. We first use $\Phi^{\mathrm{V}} \cdot x$ in Proposition 3.6 to recover a vector $y \in \mathbb{F}_p^n$; then, since we know Φ^{R}, we can also compute $\Phi^{\mathrm{R}} \cdot y$ and use Proposition 3.7 to check whether $\Phi^{\mathrm{R}} \cdot y = \Phi^{\mathrm{R}} \cdot x$: if yes, we output y and otherwise output 'fail'. It is easy to see that if x is indeed k-sparse, this scheme always recovers x correctly, and in any other case, w.h.p., it does not recover a wrong vector (but may output 'fail').

3.3 Palette Graphs, Matching View of Coloring, and Random Graph Theory

We also borrow a key technique from [7] for coloring almost-cliques in the decomposition. In the following, we shall follow the presentation of [7] as specified in the notes by [1] which is conceptually identical but notation-wise slightly different from the original presentation.

We note that this subsection might be rather too technical and not intuitive enough at this stage and can be skipped by the reader on the first read of the paper—we will get to these topics only starting from Section 5.2 of our algorithm once we start with the final coloring step of our algorithm, and by that time we have set the stage more for these definitions.

DEFINITION 3.8. Let $G = (V, E)$ be a graph, $q \geqslant 1$ be an integer, and C be a proper partial q-coloring of G. Let K be any almost-clique in G. We define the **base palette graph** of K and C as the following bipartite graph $\mathcal{G}_{\mathsf{Base}} := (\mathcal{L}, \mathcal{R}, \mathcal{E}_{\mathsf{Base}})$:

— *Vertex-set:* $\mathcal{L}$ consists of all vertices in K that are uncolored by C and $\mathcal{R}$ consists of all the colors in $[q]$ that are not assigned to any vertex in K. To avoid ambiguity, we use *nodes* to refer to elements of $\mathcal{L}$ and $\mathcal{R}$ (as opposed to vertices), and call nodes in $\mathcal{L}$ as *vertex-nodes*, and nodes in $\mathcal{R}$ as *color-nodes*.

— *Edge-set:* there is an edge between any pair of vertex-node $v \in \mathcal{L}$ and color-node $c \in \mathcal{R}$ iff $c \in \mathsf{Avail}_C(v)$, i.e., c does not appear in the neighborhood of v in the partial coloring C.

Figure 6 gives an illustration of this definition.

The base palette graph gives us a different graph theoretic way of looking at graph coloring. Suppose we start with a proper partial q-coloring C of G and manage to find an **$\mathcal{L}$-perfect matching** $\mathcal{M}$ in $\mathcal{G}_{\mathsf{Base}}$;[6] this will allow us to find an extension of C that colors all vertices of K: simply color each vertex $v \in K$ with the color c which corresponds to the matched pair of $\mathcal{M}(v)$ (in $\mathcal{G}_{\mathsf{Base}}$). This "matching view" of the coloring problem turns out to be quite helpful when analyzing almost-cliques in [7], and we shall use and considerably generalize this idea in this paper as well.

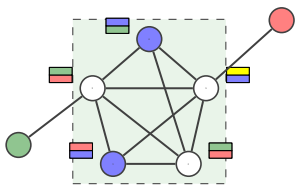
There is however an obvious obstacle in using $\mathcal{G}_{\mathsf{Base}}$ when coloring the graph: we may not have access to all of $\mathcal{G}_{\mathsf{Base}}$ (when using a semi-streaming algorithm due to space limitations). This motivates the next definition.

DEFINITION 3.9. Let $G = (V, E)$ be a graph, $q \geqslant 1$ be an integer, and C be a proper partial q-coloring of G. Let K be any almost-clique in G and $\mathcal{S} := \{S(v) \subseteq [q] \mid v \in K\}$ be a collection of sampled colors.[7] We define the **sampled palette graph** of K, C, and $\mathcal{S}$, denoted by $\mathcal{G}_{\mathsf{Sample}} = (\mathcal{L}, \mathcal{R}, \mathcal{E}_{\mathsf{Sample}})$ as the spanning subgraph of the base palette graph $\mathcal{G}_{\mathsf{Base}}$ obtained by letting $\mathcal{E}_{\mathsf{Sample}}$ to be the edges (v, c) from $\mathcal{E}_{\mathsf{Base}}$ such that $c \in S(v)$.

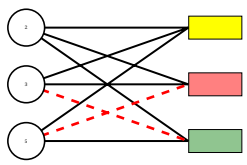
Let us again fix a small almost-clique K and the partial coloring C. Consider the sampled palette graph $\mathcal{G}_{\mathsf{Sample}}$ of K, C, and $\mathcal{S} := \{S(v) \mid v \in K\}$ for $S(v)$ of size, say, polylog (n), chosen randomly from $[q]$. This is now a much sparser subgraph that is easier to maintain via a

6 Clearly, this will not be always possible, for instance when size of K is larger than q—in general, one needs some preprocessing steps before being able to apply this idea.

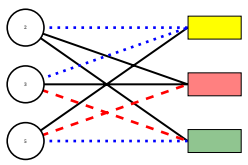
7 Think of $S(v)$ as being a small set of colors sampled for each vertex. For instance, in the context of palette sparsification theorem of [7], $S(v)$ is the set of $O(\log n)$ colors sampled for each vertex.



(a) an almost-clique



(b) base palette graph (dashed lines are missing edges)



(c) sampled palette graph (dotted lines are not-sampled edges)

Figure 6. An illustration of base palette graphs (Definition 3.8) and sampled palette graphs (Definition 3.9).

semi-streaming algorithm. Similar to before, if we find an $\mathcal{L}$-perfect matching in $\mathcal{G}_{\texttt{Sample}}$ we will be done. The challenge now, however, is to argue that not only $\mathcal{G}_{\texttt{Base}}$, but even $\mathcal{G}_{\texttt{Sample}}$ (that has much fewer edges) contains such a matching (with high probability over the choice of sampled lists).

This challenge can be addressed using "random graph theory type" arguments: we first establish several key properties of $\mathcal{G}_{\texttt{Base}}$ itself that ensure that it has an $\mathcal{L}$-perfect matching; then, we consider $\mathcal{G}_{\texttt{Sample}}$ which is a random subgraph of $\mathcal{G}_{\texttt{Base}}$ and use simple tools in the analysis of random graphs to prove that $\mathcal{G}_{\texttt{Sample}}$ also w.h.p. has an $\mathcal{L}$-perfect matching.[8] The following lemma mentions one example of such a random graph theory type argument that played a key role in [7] (we shall use this lemma and some news ones that we establish throughout our proofs in this paper).

LEMMA 3.10 ([7]). *Let $H = (L, R, E)$ be any bipartite graph with the following properties:*

- (i) *$m := |L|$ and $m \leqslant |R| \leqslant 2m$;*
- (ii) *The minimum degree of vertices in L is at least $(2/3) \cdot m$, i.e., $\min_{v \in L} \deg_H(v) \geqslant (2/3) \cdot m$;*
- (iii) *For every set $A \subseteq L$ of size $|A| \geqslant m/2$, we have $\sum_{v \in A} \deg_H(v) \geqslant (|A| \cdot m) - m/4$.*

For any $\delta \in (0, 1)$, a subgraph of H obtained by sampling each edge independently with probability at least $(\frac{20}{m} \cdot (\log m + \log(1/\delta)))$ contains an L-perfect matching with probability at least $1 - \delta$.

As discussed earlier, the way one applies Lemma 3.10 is by setting H to be the base palette graph of a given almost-clique, in which case, sampled palette graph has the same distribution as specified in Lemma 3.10 and thus w.h.p. will have the desired matching. Also, while at this stage the properties of H in this lemma may sound rather arbitrary, as we shall see later in our proofs, they appear naturally as properties of base graphs of (certain) almost-cliques. Finally, we note that even though the proof of this lemma is rather technical and so we do not repeat

8 Note that if $\mathcal{G}_{\texttt{Base}}$ is indeed a bipartite clique, then $\mathcal{G}_{\texttt{Sample}}$ would become a standard random graph. However, in general, $\mathcal{G}_{\texttt{Base}}$ can be "sufficiently far" from a bipartite clique, which requires a careful analysis of $\mathcal{G}_{\texttt{Sample}}$ beyond known results in random graph theory. See Lemma 3.10 (and its proof in [7]) for an example.

it here, it is not hard to verify that at least the graph H itself has an L-perfect matching using Hall's theorem (Fact 3.1), given the conditions imposed on it in Lemma 3.10.

4. The Semi-Streaming Algorithm

In this section we describe the algorithm that collects necessary information for the Δ-coloring from the stream. This will then be used in our main coloring procedure in Section 5 to prove Theorem 1.1. The algorithm consists of the following three main parts:

— The `palette-sampling` algorithm (Algorithm 1): An algorithm, quite similar to palette sparsification approach, that samples polylog (n) potential colors for each vertex and store all possibly monochromatic edges during the stream.

— The `find-decomposition` algorithm (Algorithm 2): An algorithm that recovers a sparse-dense decomposition of the input graph as specified in Proposition 3.5 plus some extra useful information about the decomposition.

— The `sparse-recovery` algorithm (Algorithm 3): An algorithm that uses sparse recovery techniques to extract further "helper structures" about the almost-cliques in the decomposition of the previous step.

We elaborate on each of these algorithms and their guarantees in the following subsections. But we shall emphasize that they all run *in parallel* in a single pass over the stream. To continue, we start with setting up some parameters and key definitions.

4.1 Parameters, Classification of Almost-Cliques, and Helper Structures

We use the following parameters for the design of our algorithms.

$$\begin{aligned} &\alpha = 10^3 : && \text{a large constant used to simplify various concentration inequalities} \\ &\beta = 100 \cdot \log n : && \text{used to bound the size of certain palettes in } \texttt{palette-sampling} \\ &\varepsilon = \frac{10^{-8}}{\log n} : && \text{used as the parameter of sparse-dense decomposition of Proposition 3.5.} \end{aligned} \tag{1}$$

We also assume[9] that $\Delta = \Omega(\log^5 n)$ as otherwise we can simply store the graph entirely and solve the problem offline, using any classical algorithm for Brooks' theorem.

Recall the notion of almost-cliques in Definition 3.4 used in our sparse-dense decomposition. In the coloring phase of the algorithm, we make further distinctions between almost-cliques based on their sizes as defined below—the coloring algorithm will treat these classes separately and our algorithms in this section provide further information about these different classes.

9 This is used in the proof of Lemma 5.2.

Classification of almost-cliques. We start with the following simple definition that partitions almost-cliques based on their size.

DEFINITION 4.1. Let K be an almost-clique in the sparse-dense decomposition. We say that K is **small** iff it has at most Δ vertices, **critical** iff it has exactly $\Delta+1$ vertices, and **large** otherwise.

We have the following basic observation based on this definition.

OBSERVATION 4.2. (i) *Any critical almost-clique contains at least one non-edge, and* (ii) *any large almost-clique contains at least* $(\Delta+2)/2$ *non-edges.*

PROOF. Property (i) holds because there are no $(\Delta+1)$ cliques in our input as otherwise the graph will not be Δ-colorable, and (ii) holds because maximum degree of vertices is Δ and thus every vertex has at least one non-edge in a large almost-clique. ■

A property that fundamentally affects how we color an almost-clique is the number of non-edges inside it. Intuitively, if an almost-clique is very "clique-like", that is, it has very few non-edges inside, then it is more difficult to color. This motivates the following definition.

DEFINITION 4.3. Let K be an almost-clique in the sparse-dense decomposition. We say that K is **holey** iff it has at least $10^7 \cdot \varepsilon\Delta$ non-edges (or "holes") inside it. Otherwise, K is **unholey**.

Another key property that governs our ability to color an almost-clique is how it is connected to the outside and in particular, what we can expect from a coloring of its neighbors outside: can we see those colors as being "random" or are they "adversarial"? In the latter case, can we recolor some to make them "less adversarial"? This motivates the following two definitions.

DEFINITION 4.4. Let K be an almost-clique in the decomposition and v be any vertex outside K that is neighbor to K. We say that:

- v is a **friend** of K iff there are at least $2\Delta/\beta$ edges from v to K, i.e., $|N(v) \cap K| \geqslant 2\Delta/\beta$;
- v is a **stranger** to K iff there are less than Δ/β edges from v to K, i.e., $|N(v) \cap K| < \Delta/\beta$;

We emphasize that there is a gap in the criteria between friend and stranger vertices, and hence it is possible that a vertex is neither a friend of nor a stranger to an almost-clique. Based on the notion of friend and stranger vertices, we can further classify almost-cliques into these classes.

DEFINITION 4.5. Let K be an almost-clique in the decomposition. We say that K is:

- **friendly** iff K has at least one neighbor outside K that is a friend of K.
- **lonely** iff all neighbors of K outside K are strangers.

— **social** otherwise; that is, K has at least one neighbor outside K that is *not* a stranger, but at the same time, it has no friends.

We approach coloring friendly and lonely almost-cliques quite differently, but both approaches can handle social almost-cliques. This will be crucial as we can distinguish between friendly and lonely almost-cliques but our tester may classify social almost-cliques in either of these groups.

Helper structures. As stated earlier, the problematic almost-cliques to Δ-color are unholey ones. Some of these unholey almost-cliques can be handled by a "global" argument that reason about the coloring of their neighbors outside. But for the rest, we may need to consider recoloring some of their neighbors outside and/or using some extra information about the graph. In the following, we define two "helper structures" that provide this extra information for our coloring approach. Our algorithms in this section then show how we can find these subgraphs in the stream.

The first structure we have handles unholey almost-cliques which are critical.

DEFINITION 4.6. Let K be an unholey almost-clique which is critical. We define a **critical-helper structure** for K as a tuple $(u, v, N(v))$ with the following properties:

(i) u, v are vertices of the graph and are both in K;
(ii) u and v are non-neighbor to each other;
(iii) $N(v)$ is the neighborhood of v in the graph.

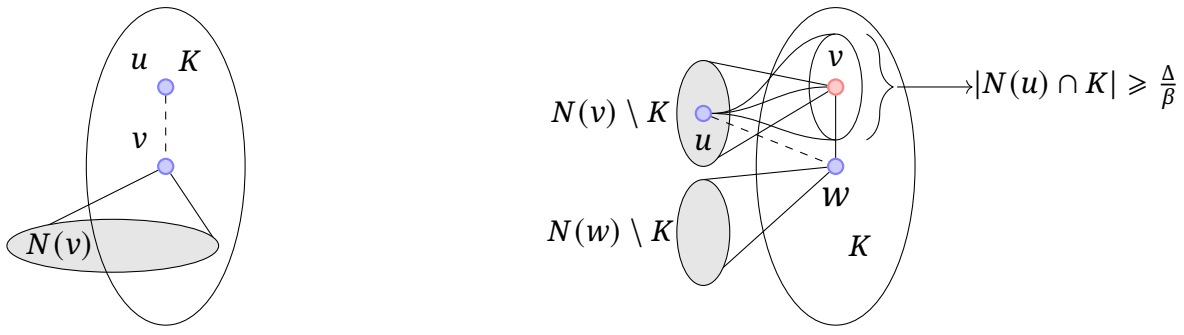
At a very high level, if we have a critical-helper structure of K at hand, we can color u and v the same (the crucial knowledge of $N(v)$ allows us to do this) which "buys" us an extra color which will be sufficient for us to color the entire almost-clique also.

The second structure we have handles unholey almost-cliques which are friendly or even social.

DEFINITION 4.7. Let K be an unholey almost-clique which is either friendly or social. We define a **friendly-helper structure** for K as a tuple $(u, v, w, N(v), N(w))$ with the following properties:

(i) u, v, w are vertices of the graph such that $u \notin K$ and is <u>not</u> a stranger to K and $v, w \in K$;
(ii) u is neighbor to v and non-neighbor to w, and v and w are themselves neighbors;
(iii) $N(v)$ and $N(w)$ are the neighborhoods of v and w in the graph, respectively.

Again, at a high level, if we have a friendly-helper structure of K at hand, we will be able to (re)color u and w the same, which "buys" us an extra color for v and gives us the required flexibility for coloring the entire almost-clique (the knowledge of $N(v)$ and $N(w)$ crucially



(a) A critical helper: The vertices u and v can receive the same color.

(b) A friendly helper: The vertices u and w can receive the same color.

Figure 7. The two types of helper structures in Definition 4.6 and Definition 4.7.

allows us to choose the colors for these vertices without creating a conflict with their neighbors in the graph).

4.2 Palette Sampling

One key component of our algorithm is a color sampling procedure in the same spirit as the palette sparsification theorem of [7].

Input: Graph $G = (V, E)$ with known vertices V and streaming edges E.

(i) For every vertex $v \in V$, sample the following lists of colors:
- $L_1(v)$: Sample a single color chosen uniformly at random from $[\Delta]$.
- $L_2(v)$: Sample each color in $[\Delta]$ independently with probability $\frac{\beta}{\Delta}$.
- $L_3(v)$: Sample each color in $[\Delta]$ independently with probability $\frac{100 \cdot \alpha \cdot \log n}{\varepsilon^2 \cdot \Delta}$.
- $L_4(v) := (L_4^*(v)$ and $L_{4,i}(v)$: $i \in [\beta])$: Independently, sample each color in $[\Delta]$ with probability $\frac{\beta}{\Delta}$ in $L_4^*(v)$ and with $q := \frac{1}{100\sqrt{\varepsilon}\Delta}$ in $L_{4,i}(v)$ for $i \in [\beta]$.
- $L_5(v)$: Sample each color in $[\Delta]$ independently with probability $\frac{\beta}{\Delta}$.
- $L_6(v) := (L_{6,i}(v)$: $i \in [2\beta])$: Sample each color in $[\Delta]$ independently with probability β^2/Δ.

(ii) Let $L(v) := \cup_{j \in [6]} L_j(v)$. Store any edge (u, v) if $L(u) \cap L(v) \neq \emptyset$ and let H be the subgraph of G on these stored edges, referred to as the **conflict graph**.

Algorithm 1. The `palette-sampling` algorithm.

The main difference of this algorithm with that of [7] is that the number of sampled colors per vertex is larger here (polylog (n) as opposed to $O(\log n)$) and that we explicitly partition these samples into multiple lists instead of just one.

At this point, the choices of the lists $L_1(v) \dots L_6(v)$ may seem arbitrary:[10] the (very) rough idea is that we will use different lists at different stages of the coloring, and the sizes are chosen to allow each stage to go through without causing too much dependency for the next stage. In the rest of this subsection, we will bound the space complexity of `palette-sampling` (Algorithm 1).

Recall that in the conflict graph G, we only keep an edge (u, v) if $L(u) \cap L(v) \neq \emptyset$—which makes sense, since if we restrict ourselves to coloring vertices with a color from their lists $L(\cdot)$, then these are the only edges that can be monochromatic. We have the following (standard) lemma.

LEMMA 4.8. *With high probability, conflict graph H in* `palette-sampling` *has $O(n \cdot \log^6 n)$ edges.*

PROOF. First, note that for any vertex $v \in V$, $L_1(v)$ has a single color. We want to establish that the sizes of the other lists $|L_2(v)|, \dots, |L_6(v)|$ are bounded by $O(\log^3 n)$ with high probability.

For $L_2(v)$: We have that the expected size of $L_2(v)$ is $\frac{\beta}{\Delta} \cdot \Delta = \beta$ by linearity of expectation. Since each color is sampled into $L_2(v)$ independently, we have via an application of Chernoff bound (Proposition 3.2, with $\delta = 1$) that:

$$\Pr\left(|L_2(v)| > 2\beta\right) \leqslant \exp\left(-\frac{1^2 \cdot \beta}{3+1}\right) \leqslant n^{-25}.$$

By union bound, we have that with high probability, $|L_2(v)|$ is bounded by $2\beta = O(\log n)$ for all $v \in V$. We can apply the same argument on all lists with expected size $\Omega(\log n)$ to show that their sizes are within a constant of their respective expected values with high probability. In particular, $L_3(v)$, $L_4^*(v)$, $L_5(v)$, all have expected sizes that are $\Omega(\log n)$ and also $O(\log^3 n)$ (see Eq. (1)), and hence have size $O(\log^3 n)$ for each one with high probability. Further, since for each i, $L_{6,i}(v)$ has expected size β^2, $|L_6(v)|$ is $O(\log^3 n)$ as well, with high probability.

This leaves us with the lists $L_{4,i}(v)$: Note that their expected size is $\frac{1}{100\sqrt{\varepsilon}} = 100\sqrt{\log n}$. Since each color is sampled into $L_{4,i}(v)$ independently, we can use a Chernoff bound with $\delta = 10\sqrt{\log n}$ to get:

$$\Pr\left(\left|L_{4,i}(v)\right| > 1000 \log n\right) \leqslant \exp\left(-\frac{100 \log n \cdot 100\sqrt{\log n}}{3 + 10\sqrt{\log n}}\right) \leqslant n^{-100}.$$

And hence w.h.p., $|L_4(v)|$ is $O(\log^2 n)$.

At this point, we have that there exist an absolute constant $\gamma > 0$ such that with high probability, $|L(v)| < \gamma \cdot \log^3 n$ for all $v \in V$. We condition on this event for the rest of this proof.

Our strategy is to bound $\deg_H(u)$ for each $u \in V$. Recall that an edge $\{u, v\}$ is in H if $L(v)$ samples a color from $L(u)$. An arbitrary color $c \in [\Delta]$ is in $L(v)$ with probability at

10 And indeed redundant; technically speaking, we could have just sampled a single list of colors of proper size and postponed the partitioning to the analysis (as in fact done in [7])—however, we find it more transparent to consider these lists explicitly separate from each other due to various dependency issues that this explicitly avoids.

most $\gamma \cdot \log^3(n)/\Delta$ (since each of the lists $L_i(v)$ is sampled uniformly). Then for any choice of $L(u)$ (of size at most $\gamma \cdot \log^3 n$), we have by the union bound:

$$\Pr\left(L(u) \cap L(v) \neq \emptyset\right) = \Pr\left(\bigcup_{z \in L(u)} z \in L(v)\right) \leqslant \sum_{z \in L(u)} \Pr\left(z \in L(v)\right) = \frac{\gamma^2 \log^6(n)}{\Delta},$$

where the randomness is over the choice of $L(v)$.

Now we can bound the degree of an arbitrary vertex u in the subgraph H formed by the edges stored in Algorithm 1. First, fix the list $L(u)$—we already conditioned on the event that it is small, but now we will "give up" the remaining randomness in the choice of $L(u)$, and proceed by assuming an arbitrary choice. Let $X_{u,v}$ be the indicator random variable that is 1 iff the edge $\{u, v\}$ is in H. Then the expected degree of u is $(\gamma^2 \cdot \log^6 n)$ by the previous argument and linearity of expectation. Finally, we observe that for $v \neq w$, $X_{u,v}$ and $X_{u,w}$ are independent, and via another application of Chernoff bound (Proposition 3.2, with $\delta = 1$), we have:

$$\Pr\left(\deg_H(u) \geqslant 2\gamma^2 \log^6 n\right) \leqslant \exp\left(-\frac{\gamma^2 \log^6 n}{4}\right).$$

Thus by union bound, each $u \in V$ has degree $O(\log^6 n)$ in H with high probability. This immediately implies the bound on the number of edges stored by Algorithm 1. ∎

We can now bound the space used by `palette-sampling`. We have done all of the heavy-lifting already, by bounding the number of edges stored in H. The only new observation in the following lemma is that edges and colors can be stored in $O(\log n)$ bits.

LEMMA 4.9. *With high probability,* `palette-sampling` *uses* $O(n \cdot \log^7 n)$ *bits of space.*

PROOF. We showed in the proof of Lemma 4.8 that with high probability, $|L(v)| = O(\log^3 n)$ for all $v \in V$. Since each color is from $[\Delta]$, it can be represented by $\lceil \log \Delta \rceil$ bits, and hence storing the lists $L(v)$ for all v uses $O(n \log^4 n)$ bits.

Further, each edge can be represented by $2 \cdot \lceil \log n \rceil$ bits, so by invoking Lemma 4.8, we have that we need $O(n \cdot \log^7 n)$ bits to store the conflict graph H. ∎

4.3 Finding the Decomposition

We also work with the sparse-dense decomposition, but unlike [7], not only as an analytical tool but in fact algorithmically (as will become evident from the next subsection). We now describe an algorithm for finding the sparse-dense decomposition of Proposition 3.5. In particular, we only need to provide the random edge and vertex samples required by the proposition. But, in addition to the samples required for the decomposition, we will also collect independent random edge samples to allow us to distinguish friend vertices from strangers (Definition 4.4).

Let us start by bounding the space used by `find-decomposition` and then present the main properties of the algorithm for our purpose.

Input: Graph $G = (V, E)$ with known vertices V and streaming edges E.

(*i*) Let γ be the constant from the statement of Proposition 3.5.

(*ii*) *Vertex samples:* For each vertex v, sample v into the set SAMPLE with probability $(\gamma \cdot \log n/\Delta)$ independently. During the stream, for each vertex v in SAMPLE, store all edges incident on v.

(*iii*) *Edge samples:* For each vertex v, use reservoir sampling on edges of v to pick a sample $N_{\text{sample}}(v)$ of size $\left(\gamma \cdot \varepsilon^{-2} \cdot \log n\right)$ from its neighborhood.

(*iv*) *Neighbor samples:* For each vertex v, store each neighbor of v in $I_{\text{sample}}(v)$ with probability β^2/Δ.

Algorithm 2. The `find-decomposition` algorithm.

LEMMA 4.10. *With high probability,* `find-decomposition` *uses* $O(n \log^4 n)$ *bits of space.*

PROOF. The set SAMPLE has size $\gamma \cdot n \log n/\Delta$ in expectation, and since each vertex v is in SAMPLE independently, the size is at most (say) $5\gamma \cdot n \log n/\Delta$ with high probability by Chernoff bound (Proposition 3.2). For each $v \in$ SAMPLE, we use upto $\Delta \cdot \lceil \log n \rceil$ bits of space to store all of its edges, and hence in total we use $O(n \log^2 n)$ bits to store SAMPLE and the neighborhood of vertices in it.

The sets $N_{\text{sample}}(v)$ have fixed sizes $(\gamma \cdot \varepsilon^{-2} \cdot \log n)$ each. Storing a neighbor takes $\lceil \log n \rceil$ bits, and hence storing all the sets $N_{\text{sample}}(v)$ takes $O(n \log^4 n)$ bits of space.

The set $I_{\text{sample}}(v)$ is of size at most $2\beta^2$ with high probability (again, the proof is the same as in Lemma 4.8). Hence storing $I_{\text{sample}}(v)$ for all v uses $O(n \log^3 n)$ bits of space. ■

We now establish the main properties we need from `find-decomposition`.

LEMMA 4.11. *We can compute a sparse-dense decomposition (Proposition 3.5) of the input graph* $G = (V, E)$ *with high probability using the samples collected by* `find-decomposition`*.*

PROOF. The proof is immediate—we collect SAMPLE and $N_{\text{sample}}(v)$ as needed by Proposition 3.5, so we can use the algorithm in the proposition to compute the decomposition. ■

We will also show that the independent random edge samples $I_{\text{sample}}(v)$ are enough for a tester that can distinguish friends from strangers for any almost-clique.

LEMMA 4.12. *There exists an algorithm that given an almost-clique* K*, a vertex* $v \notin K$*, and the random neighbor samples* $I_{\text{sample}}(v)$*, with high probability can distinguish:*

— *v has at most Δ/β neighbors in K, that is, v is a stranger to K;*

— *v has at least $2\Delta/\beta$ neighbors in K, that is, v is a friend of K.*

The randomness in this lemma is only over the sample $I_{\text{sample}}(v)$.

PROOF. Fix a vertex $v \notin K$. Let u be any neighbor of v in K. Then $u \in I_{\text{sample}}(v)$ with probability β^2/Δ. Let $X_u \in \{0,1\}$ be the indicator random variable which is 1 iff $u \in I_{\text{sample}}(v)$. Thus, $X := \sum_{u \in N(v) \cap K} X_u$ counts the size of intersection of $I_{\text{sample}}(v)$ with K. Firstly, we have

$$\mathbb{E}[X] = \sum_{u \in N(v) \cap K} \frac{\beta^2}{\Delta} = |N(v) \cap K| \cdot \frac{\beta^2}{\Delta}.$$

Thus, by Definition 4.4, $\mathbb{E}[X]$ is at least 2β if v is a friend of K and at most β if v is a stranger. Our tester can simply compute the value of X and output *friend* if X is more than $\frac{3}{2}\beta$, and *stranger* otherwise. To prove the correctness, suppose that $|N(v) \cap K| < \Delta/\beta$, i.e. v is a stranger. Then, by an application of Chernoff bound (Proposition 3.2 with $\delta = 1/2$) we have:

$$\Pr\left(X > \frac{3}{2} \cdot \beta\right) < \exp\left(-\frac{1/4 \cdot \beta}{3 + 1/2}\right) < \exp\left(-\frac{\beta}{16}\right) < n^{-6},$$

by the choice of β in Eq. (1). The other case can be proven symmetrically. Hence if v is a stranger (resp. friend), our tester also outputs stranger (resp. friend) with high probability. ■

An immediate consequence of Lemma 4.12 is that we have a tester that can distinguish friendly almost-cliques from lonely almost-cliques (Definition 4.5).

LEMMA 4.13. *There exists an algorithm that given an almost-clique K, and to the random neighbor samples $I_{\text{sample}}(v)$ for every $v \in V$, with high probability can distinguish:*

— *K is a friendly almost-clique;*

— *K is a lonely almost-clique.*

The randomness in this lemma is only over the samples $\{I_{\text{sample}}(v) \mid v \in V\}$.

PROOF. For each vertex $v \in V \setminus K$, run the tester from Lemma 4.12. If K has even one friend, then that friend is distinguished by the tester in Lemma 4.12 as a friend, and we can return that K is friendly. Otherwise, if K is lonely, it means that every vertex v we tested for K will be considered stranger also with high probability and thus we can correctly mark K as lonely. ■

4.4 Sparse Recovery for Almost-Cliques

Finally, we come to the most novel part of this section. Recall that as discussed earlier, palette sparsification (and thus our own `palette-sampling`) is doomed to fail for Δ-coloring. To bypass this, we rely on the *helper structures* defined in Definitions 4.6 and 4.7, which, combined with the palette sparsification-type approach of `palette-sampling`, allow us to color the graph.

The first challenge here is that we obviously cannot afford to find the neighborhood of every vertex, and we do not know *during* the stream which vertices will satisfy the properties

we need for these structures. We step around this by (crucially) using randomization to sample the "right" vertices. The second and main challenge is that for some almost-cliques (say, a critical almost-clique with only one non-edge), we may actually have to recover neighborhood of *all* vertices in the almost-clique before finding the required helper structure; but doing this naively requires too much space. Instead, we use the sparse recovery matrices of Section 3.2, in conjunction with the decomposition found by `find-decomposition`, to recover these parts much more efficiently.

Input: Graph $G = (V, E)$ with known vertices V and streaming edges E.
For every $r \in R = \{2^i \mid 0 \leqslant i \leqslant \lceil \log \Delta \rceil\}$:[11]

- (i) Sample each vertex $v \in V$ in a set V_r independently with probability $\min\{1, \frac{\beta}{\varepsilon \cdot r}\}$.
- (ii) Construct the $(2r \times n)$ *Vandermonde matrix* Φ_r^{V} (see Proposition 3.6) and sample an $(\alpha \times n)$ *random matrix* Φ_r^{R} (see Proposition 3.7) over the field $\mathbb{F}_p$ where p is a fixed prime larger than n and smaller than, say, n^2, and α is the parameter in Eq. (1).
- (iii) For each vertex $v \in V_r$, define a vector $y(v) \in \mathbb{F}_p^{2r}$ and $z(v) \in \mathbb{F}_p^{\alpha}$ initially set to 0. For any incoming edge $\{u, v\}$ in the stream, update

$$y(v) \leftarrow y(v) + \Phi_r^{\mathrm{V}} \cdot \mathbf{e}_u \qquad \text{and} \qquad z(v) \leftarrow z(v) + \Phi_r^{\mathrm{R}} \cdot \mathbf{e}_u,$$

  where $\mathbf{e}_u$ is the n-dimensional vector which is 1 on coordinate u and 0 everywhere else.

Algorithm 3. The `sparse-recovery` algorithm.

Let us start by bounding the space of this algorithm.

LEMMA 4.14. `sparse-recovery` *uses* $O(n \log^4 n)$ *bits of space.*

PROOF. For each $r \in R$, each $v \in V$ is sampled into V_r with probability (at most) $\frac{\beta}{\varepsilon r}$, which means the expected size of V_r is (at most) $\frac{\beta n}{\varepsilon r}$. Since each sample is independent, we have by Chernoff bound (Proposition 3.2) that,

$$\Pr\left(|V_r| > 2 \cdot \frac{\beta n}{\varepsilon r}\right) \leqslant \exp\left(-\frac{\beta n}{4\varepsilon r}\right) \leqslant \exp\left(-\frac{100 n \log n}{4\varepsilon \cdot 2\Delta}\right) \leqslant n^{25/2 \cdot \varepsilon} \ll 1/\mathrm{poly}(n),$$

by the choice of $\varepsilon = \Theta(\log^{-1}(n))$ in Eq. (1).

Combining with the union bound over $O(\log \Delta)$ choices of $r \in R$, we have that each V_r is of size at most $\frac{2\beta n}{\varepsilon r}$ with high probability. For each $v \in V_r$, Algorithm 3 stores two vectors $y(v)$ and $z(v)$ that require $O(r \cdot \log p)$ and $O(\alpha \cdot \log p)$ bits, respectively, where p is the order of the field $\mathbb{F}_p$. This, together with the bound on V_r and since $\alpha = \Theta(1)$ by Eq. (1) means that the total

number of bits needed to store these vectors is $O(\varepsilon^{-1} \cdot \beta n \log p \cdot \log \Delta) = O(n \log^4 n)$ bits (where we used the fact that both ε^{-1} and $\log p$ are bounded by $O(\log n)$).

Finally, the algorithm does not need to explicitly store the Vandermonde matrix for Φ_r^{V} (as each of its entries can be easily generated in $O(\log p)$ space at any point) and can store each random matrix Φ_r^{V} in $O(\alpha \cdot n \log p) = O(n \log n)$ bits as $\alpha = \Theta(1)$ and $\log p = O(\log n)$. Thus, for all $r \in R$, we also need to store $O(n \log^2 n)$ bits to store the random matrices. Overall, the total space of the algorithm is still $O(n \log^4 (n))$, concluding the proof. ■

We now switch to proving the main properties of `sparse-recovery` for our purpose. Before getting into details however, we state a standard observation about linear transformations (namely, Φ_r^{V} and Φ_r^{R} in `sparse-recovery`) over a stream of updates.

OBSERVATION 4.15. *Fix any $r \in R$ and $v \in V_r$ in* `sparse-recovery`*. At the end of the stream,*

$$y(v) = \Phi_r^{\mathrm{V}} \cdot \chi(N(v)) \qquad \text{and} \qquad z(v) = \Phi_r^{\mathrm{R}} \cdot \chi(N(v)),$$

where $\chi(N(v)) \in \{0,1\}^V$ is the characteristic vector of $N(v)$.

PROOF. We only prove the equation for $y(v)$; the one for $z(v)$ follows similarly. Initially, we set $y(v) = 0$. Then, during the stream, whenever we see the edge $\{u, v\}$ for each $u \in N(v)$, we update $y(v)$ by adding $\Phi_r^{\mathrm{V}} \cdot \mathbf{e}_u$ to it. As Φ_r^{V} is a linear transformation, we have,

$$y(v) = \sum_{u \in N(v)} \Phi_r^{\mathrm{V}} \cdot \mathbf{e}_u = \Phi_r^{\mathrm{V}} \cdot \left(\sum_{u \in N(v)} \mathbf{e}_u \right) = \Phi_r^{\mathrm{V}} \cdot \chi(N(v)),$$

concluding the proof. ■

Recall from Proposition 3.6 that given $\Phi_r^{\mathrm{V}} \cdot x$ for an r-sparse vector $x \in \mathbb{F}_p^n$, we can recover x in polynomial time, and by Proposition 3.7, we can test our recovered vector to make sure with high probability that it is indeed equal to x. We use this idea combined with the fact that at the end of the stream we know a sparse-dense decomposition of the graph to recover one helper structure for each almost-clique (that needs one). We start with the key part that handles the critical-helper structures (Definition 4.6). There will also be a simpler part that handles friendly-helper structures almost-cliques (Definition 4.7).

Finding Critical-Helper Structures. We start by recovering a critical-helper structure for any critical almost-clique as defined in Definition 4.6. Let K be a critical almost-clique and v be a vertex in K. Define the vector $x(v) := \chi(N(v)) - \chi(K)$. For any coordinate $u \in [n]$ in $x(v)$,

$$x(v)_u = \begin{cases} 0 & \text{if } u \notin N(v) \cup K \text{ or } u \in N(v) \cap K \\ 1 & \text{if } u \in N(v) \setminus K \\ p-1 & \text{if } u \in K \setminus N(v) \end{cases}, \tag{2}$$

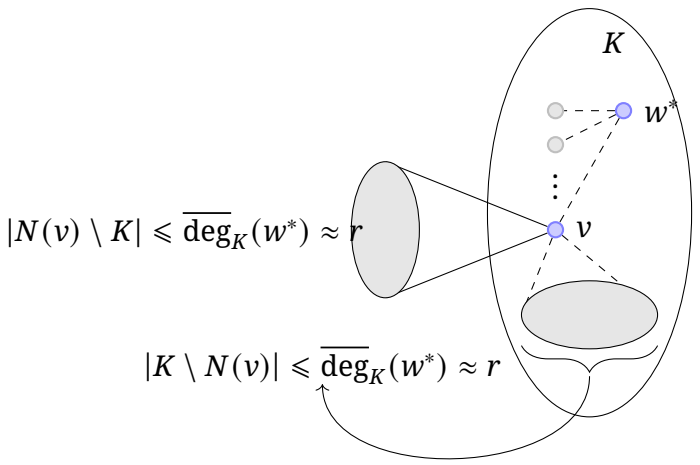


Figure 8. Finding a critical-helper: With high probability, there is a vertex $v \in \overline{N}_K(w^*)$ which is sampled in V_r.

as we do the computation over $\mathbb{F}_p$. Notice that since v belongs to the almost-clique K, by Definition 3.4, size of both $N(v) \setminus K$ and $K \setminus N(v)$ is at most $10\varepsilon\Delta$. Thus, $x(v)$ is already considerably sparser than $\chi(N(v))$. In the following, we are going to take this idea to the next level to recover a critical-helper structure for K using the vectors computed by `sparse-recovery`.

LEMMA 4.16. *There exists an algorithm that given a critical almost-clique K (Definition 4.1), with high probability, finds a critical-helper structure $(v, u, N(v))$ of K (Definition 4.6) using the information gathered by* `sparse-recovery`.

PROOF. (Sidefigure 8 gives an illustration that might be helpful to refer to during the proof.) For any vertex $w \in K$, define:

— $\overline{N}_K(w) := K \setminus N(w)$: as the non-edge neighborhood of w in K and $\overline{\deg}_K(w) = \left|\overline{N}_K(w)\right|$ as the non-edge degree of w.

Define w^* as the vertex that maximizes this non-edge degree, i.e., $w^* = \arg\max_{w \in K} \overline{\deg}_K(w)$. By Observation 4.2, we have $\overline{\deg}_K(w^*) \geqslant 1$. We first have the following simple claim that will be crucial in finding the neighborhood of at least one vertex in $\overline{N}_K(w^*)$ using sparse recovery.

CLAIM 4.17. *Let $r \in R$ be the smallest integer such that $r \geqslant 2 \cdot \overline{\deg}_K(w^*)$. Then, for every $w \in \overline{N}_K(w^*)$, the vector $x(w) := \chi(N(w)) - \chi(K)$ is r-sparse.*

Proof. Fix any $w \in \overline{N}_K(w^*)$. By the definition of w^*, we have $\overline{\deg}_K(w) \leqslant \overline{\deg}_K(w^*) \leqslant r/2$. At the same time, since K is a critical almost-clique and thus has size $\Delta + 1$, this means that the number of neighbors of w outside K is also at most $\overline{\deg}_K(w^*) \leqslant r/2$. By Eq. (2), this means that $x(w)$ has at most r non-zero entries. ◆

Consider the parameter r of Claim 4.17. We have that $\overline{\deg}_K(w^*) \geqslant r/4$ as elements of R are within a factor two of each other and by the value of r. Given that each vertex is chosen in r with probability $\min\{1, \beta/(\varepsilon \cdot r)\}$, we have,

$$\Pr\left(V_r \cap \overline{N}_K(w^*) = \emptyset\right) \leqslant \left(1 - \frac{\beta}{\varepsilon \cdot r}\right)^{r/4} \leqslant \exp\left(-\frac{\beta}{4\varepsilon}\right) \ll 1/\text{poly}(n),$$

by the choice of $\beta = \Theta(\log n)$ and $\varepsilon = \Theta(\log^{-1}(n))$ in Eq. (1). In the following, we condition on the high probability event that a vertex $v \in \overline{N}_K(w^*)$ is sampled in V_r. For now, let us *assume* that we know the identity of this vertex v in V_r.

Firstly, by Claim 4.17, we have that the vector $x(v)$ is r-sparse. Secondly, since $v \in V_r$, `sparse-recovery` has computed $y(v) = \Phi^{\mathrm{V}}_r \cdot \chi(N(v))$ by Observation 4.15 and since we are given K, we can also compute $\Phi^{\mathrm{V}}_r \cdot \chi(K)$. Thus, by linearity, we can compute $\Phi^{\mathrm{V}}_r \cdot x(v)$ this way. Finally, since $x(v)$ is r-sparse, by Proposition 3.6, we can actually recover $x(v)$ from $\Phi^{\mathrm{V}}_r \cdot x(v)$. But again, since we know $\chi(K)$, this gives us $\chi(N(v))$ and in turn $N(v)$ as well. Thus, the algorithm can return the critical-helper structure $(v, u, N(v))$ for $u = w^*$ which satisfies all the properties (as (v, w^*) is a non-edge).

It only remains the remove the assumption on the knowledge of identity of v in V_r (and possibly the value of r itself). For this, we simply iterate over all vertices $w \in K$ and for each one run $\Phi^{\mathrm{V}}_r \cdot x(w)$ and $\Phi^{\mathrm{R}}_r \cdot x(w)$ as described above for all values of $r \in R$ (by using $z(v)$ in place of $y(v)$ when computing the latter). As outlined in Section 3.2, we can now apply Proposition 3.7 to the outcome of each sparse recovery to get that with high probability, any vector $\chi(N(w))$ that we recover is correct. Since we know that the vertex v will *not* output 'fail', we are guaranteed that with high probability we will return a valid critical-helper structure, concluding the proof. ■

Finding Friendly-Helper Structures. We now switch to finding a friendly-helper structure of Definition 4.7 for each almost-clique K that is unholey and *not* lonely.

LEMMA 4.18. *There exists an algorithm that given an unholey and not lonely almost-clique K (Definition 4.1) and a vertex $u \notin K$ which is not a stranger to K, with high probability, finds a friendly-helper structure $(u, v, w, N(v), N(w))$ of K (Definition 4.7) using the information gathered by* `sparse-recovery`.

PROOF. Define $r_{\max} = \max_{r \in R} r$. We have the following straightforward claim.[12]

CLAIM 4.19. *For any vertex $v \in V$, with probability at least $\frac{\beta}{2\varepsilon \cdot \Delta}$, we can recover the set $N(v)$.*

Proof. Note that $r_{\max} = 2^{\lceil \log \Delta \rceil}$ which is between Δ and 2Δ. Recall that each v is sampled into the set $V_{r_{\max}}$ with probability $\frac{\log n}{\varepsilon \cdot r_{\max}}$, which is at least $\frac{\log n}{2\varepsilon \cdot \Delta}$. And then note that for each vertex v in $V_{r_{\max}}$, `sparse-recovery` stores $\Phi^{\mathrm{V}}_{r_{\max}} \cdot \chi(N(v))$ by Observation 4.15. By Proposition 3.6, we can recover $N(v)$ from $\Phi^{\mathrm{V}}_{r_{\max}} \cdot \chi(N(v))$ for every $v \in V_{r_{\max}}$, concluding the proof of the claim. ◆

We prove Lemma 4.18 using this claim. Firstly, by property iv). of Definition 3.4, there are also at least $10\varepsilon\Delta$ vertices w in K that are not neighbors of u. By Claim 4.19, we recover $N(w)$

12 A careful reader may notice that in this claim, we actually do not really need sparse recovery; we could have simply stored all edges of vertices sampled in $V_{r_{\max}}$ explicitly during the stream. However, given that we indeed need sparse recovery for all other ranges of V_r for $r \in R$ in the previous part, we use a unified approach for $V_{r_{\max}}$ as well.

for any such choice of w with probability at least $\beta/(2\varepsilon \cdot \Delta)$. As such, we have,

$$\Pr\left(N(w) \text{ is not recovered for any } w \in K \setminus N(u)\right) \leqslant \left(1 - \frac{\beta}{2\varepsilon \cdot \Delta}\right)^{\varepsilon \cdot \Delta} \leqslant \exp\left(-\frac{\beta}{2}\right) \leqslant n^{-50},$$

by the choice of $\beta = 100 \log n$ in Eq. (1). In the following, we further condition on the high probability event that for some $w \in K \setminus N(u)$, we have recovered $N(w)$. Similar to the proof of Lemma 4.16, let us *assume* that we know the identity of w.

Now consider $N(w) \cap N(u) \cap K$; since $|N(u) \cap K| \geqslant \Delta/\beta$ as u is not a stranger to K (Definition 4.4), and $|K \setminus N(w)| \leqslant 10\varepsilon\Delta$ by property ii). of Definition 3.4, we have that

$$|N(w) \cap N(u) \cap K| \geqslant \Delta/\beta - 10\varepsilon\Delta > \Delta/2\beta$$

by the choice of parameters $\varepsilon < 10^{-6} \cdot 1/\beta$ in Eq. (1). By the same argument as above, we have,

$$\Pr\left(N(v) \text{ not recovered for any } v \in N(w) \cap N(u) \cap K\right) \leqslant \left(1 - \frac{\beta}{2\varepsilon \cdot \Delta}\right)^{\Delta/2\beta} \leqslant \exp\left(-\frac{1}{2\varepsilon}\right) \leqslant n^{-100},$$

by the choice of ε in Eq. (1). We now have: u is a vertex which is not a stranger to K, w is a non-neighbor of u in K and we have $N(w)$, and v is a neighbor of both u and w and we have $N(v)$. Thus, we can return $(u, v, w, N(v), N(w))$ as a friendly-helper structure of K.

Finally, removing the assumption on the knowledge of v and w is exactly as in the proof of Lemma 4.16: we simply go over all choices of vertices in K that we have sampled in $V_{r_{\max}}$ and check whether any pairs of them satisfy the requirements of the structure or not—by the above argument, with high probability, we will find a pair. ■

4.5 Listing the Information Collected by the Algorithm

For the ease of reference in the analysis, we now take stock of what all our algorithms collected about the graph from the stream. In particular, with high probability, we have the following information at the end of the stream:

1. A list of sampled colors $L(v)$ for every vertex $v \in V$ as specified in Algorithm 1.
   Proof: Follows from the definition of `palette-sampling` in Algorithm 1.
2. The **conflict graph** H consisting of every edge (u, v) in the graph where $L(u) \cap L(v) \neq \emptyset$.
   Proof: Follows from the definition of `palette-sampling` in Algorithm 1.
3. A decomposition of G into sparse vertices and almost-cliques as specified in Proposition 3.5.
   Proof: Follows from Lemma 4.11 for `find-decomposition` in Algorithm 2.
4. A collection $\mathcal{K}_{\text{friendly}}$ of almost-cliques that contains *all* friendly almost-cliques and *no* lonely almost-clique, and for each $K \in \mathcal{K}_{\text{friendly}}$, one vertex $u \notin K$ which is <u>not</u> a stranger to K.
   Proof: Follows from Part (3.) and Lemma 4.13 for `find-decomposition` in Algorithm 2.

5. A collection $\mathcal{K}_{\text{lonely}}$ of almost-cliques that contains *all* lonely almost-cliques and *no* friendly almost-clique. Moreover, $\mathcal{K}_{\text{friendly}} \sqcup \mathcal{K}_{\text{lonely}}$ partition *all* almost-cliques, which also implies that every social almost-clique belongs to exactly one of these two collections.
   Proof: Follows from Parts (*3.*), (*4.*), and Lemma 4.13 for `find-decomposition` in Algorithm 2.
6. A collection of $\mathcal{K}_{\text{critical}}$ of critical almost-cliques and for each $K \in \mathcal{K}_{\text{critical}}$, a **critical-helper structure** $(u, v, N(v))$ of Definition 4.6.
   Proof: Follows from Part (*3.*) and Lemma 4.16 for `sparse-recovery` in Algorithm 3.
7. A collection of **friendly-helper structures** $\{(u, v, w, N(v), N(w))\}$ of Definition 4.7, one for each $K \in \mathcal{K}_{\text{friendly}}$ such that u is the vertex specified for $K \in \mathcal{K}_{\text{friendly}}$ in Part (*4.*).
   Proof: Follows from Part (*4.*) and Lemma 4.18 for `sparse-recovery` in Algorithm 3.
8. The **recovery graph** H^+ consisting of all edges in the critical-helper structures of Part (*6.*) and in the friendly-helper structures of Part (*7.*).
   Proof: Follows from Parts (*6.*) and (*7.*).

We shall note that at this point, we covered all the process that is done by our algorithm *during* the stream and what remains is to prove this information is useful, i.e., we can indeed color the graph in the *post-processing step* using this information. This is the content of the next section.

Before moving on, we should note that, with high probability, the space complexity of (i) `palette-sampling` is $O(n \log^7 n)$ bits by Lemma 4.9, (ii) `find-decomposition` is $O(n \log^4 n)$ bits by Lemma 4.10, and (iii) `sparse-recovery` is $O(n \log^4 n)$ by Lemma 4.14. Thus, our entire streaming algorithm takes $O(n \log^7 n)$ space. This adhere to the space complexity promised in Theorem 1.1.

REMARK 4.20. As stated, the space complexity of our algorithm is bounded with high probability but not in the worst-case. This is standard to fix; simply run the algorithm as it is and whenever it attempted to use more than, say, 100 times, the space guaranteed by its expectation, terminate it and "charge" the failure probability to the error.

REMARK 4.21 (Removing Prior Knowledge of Δ). We observe that this semi-streaming algorithm does not really need to know Δ before the stream begins. In particular, we can run $O(\log n)$ independent copies of the algorithm, each with a difference "guess" of $\Delta \in \left\{2^k \mid 2^k \leqslant 2n\right\}$. At the same time, we can compute Δ at the end of the stream by simply counting for each vertex the number of edges incident to it in $O(n \log n)$ space.

An overestimate of Δ does not hurt us in terms of space usage, but an underestimate can (for example, if we guess $\Delta = 1$, and the input includes a clique on $n - 1$ vertices, the algorithm stores the entire graph). Hence, if at any point (for any guess Δ), if a vertex has degree larger than 2Δ, we stop that run of the algorithm. Now, at the end of the stream we will have:

— The actual maximum degree Δ.

— The output of the algorithm for Δ' and $2\Delta'$ such that $\Delta' \leqslant \Delta \leqslant 2\Delta'$.

But now we can get the desired samples by "resampling" the outputs from Algorithms 1 to 3. In particular, for each vertex $v \in V$, and each color $c \in L(v)$ from the run of Algorithm 1 with guess Δ', we keep c with probability Δ'/Δ. Since we are only dropping colors from the palettes, this process only removes some edges from the conflict graph. The samples of Algorithm 2 are adapted in a similar manner. Finally, the set of sampling rates R in Algorithm 3 for $2\Delta'$ is a superset of that for a (hypothetical) run with the correct guess of Δ, so we can just ignore the vectors corresponding to unused sample rates.[13]

Hence at the end of the stream we know Δ and can adapt the samples as required, so the coloring procedure in the next section can proceed as normal.

5. The Coloring Procedure

We now describe the coloring procedure that we use to find a Δ-coloring of the graph. This procedure is agnostic to the input graph, in the sense that we run it after processing the stream, and it only uses the information we gathered in the previous section, listed in Section 4.5 (throughout, we condition on the high probability event that the correct information is collected from the stream).

The general framework in our coloring procedure is the following: We will maintain a proper partial Δ-coloring $C : V \to [\Delta] \cup \{\perp\}$ (as defined in Section 3). We then go through different **phases** in the coloring algorithm and each phase updates C by coloring certain subsets of vertices, say, (a subset of) sparse vertices, or certain almost-cliques. These new colorings are typically going to be extensions of C but in certain cases, we crucially have to go back and "edit" this partial coloring, i.e., come up with a new proper partial coloring which is no longer an extension the current one. Eventually, we will color all the vertices of the graph and end up with a proper Δ-coloring.

In the following, we present the order of the phases of our coloring procedure and the task we expect from each one. Each phase shall use a different list of colors $L_1(\cdot), \cdots, L_6(\cdot)$ computed by `palette-sampling` (Algorithm 1) when updating the partial coloring. We also note that the order of running these phases is crucial as some of them present further guarantees for subsequent phases, and some of them need to assume certain properties of the current partial coloring which will no longer remain true if we change the order of phases.

On the high level, the coloring procedure is as follows (Table 1 give a summary of which combination of almost-cliques are handled in which phase).

— **Phase 1 – One-Shot Coloring (Section 5.1):** We use the single color sampled in $L_1(v)$ for every vertex $v \in V$ to color a large fraction of vertices. The effect of this coloring is that it

13 Technically, there is no need to even run Algorithm 3 separately for different guesses of Δ.

"sparsifies" the graph for sparse vertices. We note that this part is standard and appears in many other coloring results starting from, to our knowledge, [51]; see [7] for more details.

— **Phase 2 – Lonely (or Social) Small Almost-Cliques (Section 5.2):** Recall that from Part (*5.*) of Section 4.5, we have a list of $\mathcal{K}_{\text{lonely}}$ of almost-cliques that contains all lonely almost-cliques and potentially some social ones. We can easily also identify which of these almost-cliques are small (Definition 4.1) based on their size.
We will color all these small almost-cliques in $\mathcal{K}_{\text{lonely}}$ by colors in $L_2(\cdot)$. This requires a novel argument that uses the facts that: (i) these almost- cliques are "loosely connected" to outside (no "high degree" neighbor, formally friend vertices, in their neighborhood), and (ii) the coloring outside only used a limited set of colors, namely, is sampled from $L_1(\cdot)$ and $L_2(\cdot)$ so far and is thus not "too adversarial" (recall the discussion we had in Lesson 2.3 regarding necessity of such arguments). Moreover, the coloring in this phase is an extension of the last one.

— **Phase 3 – Sparse Vertices (Section 5.3):** We then conclude the coloring of all sparse vertices using the sampled lists $L_3(v)$ for every sparse vertex $v \in V$ (by Part (*3.*) of Section 4.5, we know these vertices). This part is also a standard argument as a continuation of Phase 1. But to apply this standard argument, we use the fact that even though we interleaved the standard approach with Phase 2 in the middle, since that coloring was only an extension of Phase 1 (meaning it did not *recolor* any vertex colored in Phase 1), the argument still easily goes through.
The coloring in this phase is also an extension of the last one. However, now that all sparse vertices are colored, we go ahead and remove the color of any vertex which is not sparse and nor is colored by Phase 2 (these are remnants of one-shot coloring in Phase 1 and we no longer need them now that all sparse vertices are colored). This is just to simplify the analysis for later parts.
It is worth mentioning that this interleaving of Phase 2 in the middle of Phase 1 and 3 is crucial for our arguments (this is the chicken-and-egg problem mentioned in Section 2.2): the lists $L_3(\cdot)$ used in Phase 3 are much larger than the rest and thus the "not-too-adversarial" property of coloring of outside vertices in Phase 2 will no longer be guaranteed had we changed the order of Phase 2 and 3; at the same time, changing the order of Phase 1 and 2 will also destroy the "sparsification" guarantee provided by Phase 1 for sparse vertices.

— **Phase 4 – Holey Almost-Cliques (Section 5.4):** The next step is to color holey almost-cliques (Definition 4.3), i.e., the ones with $\Omega(\varepsilon\Delta)$ non-edges inside them, using colors sampled in $L_4(\cdot)$. We note that we actually do *not* know which almost-cliques are holey and which ones are not.[14] Instead, we simply run this phase over all remaining almost-cliques and argue that all the holey ones (and possibly some other ones) will get fully colored as desired.

14 Technically, we could have designed a semi-streaming algorithm that also recovers this information about the decomposition (at least approximately). However, as we explain next, this is not needed.

Type	*Holey*			*Unholey*		
	Friendly	*Social*	*Lonely*	*Friendly*	*Social*	*Lonely*
Small	Phase 4	Phases 2 or 4	Phase 2	Phase 6	Phases 2 or 6	Phase 2
Critical	Phase 4	Phase 4	Phase 4	Phase 5	Phase 5	Phase 5
Large	Phase 4	Phase 4	Phase 4	–	–	–

Table 1. A list of all combination of different almost-cliques together with the phase of our coloring procedure that is responsible for handling them. Note that by Observation 4.2, there are no unholey large almost-cliques. The sparse vertices are handled in Phase 1 and Phase 3. Moreover, Phase 1 may color some vertices of lonely (or social) small almost-cliques that we are *not* allowed to recolor (we can recolor all the other remnants of Phase 1 after Phase 3).

The proof of this phase is a simple generalization of a similar proof used in the palette sparsification theorem of [7], which even though quite technical, does not involve much novelty from us in this work. The coloring in this phase is an extension of the last one.

— **Phase 5 – Unholey Critical Almost-Cliques (Section 5.5):** By Observation 4.2, all large almost-cliques are holey. Thus, the largest remaining almost-cliques at this point are unholey critical almost-cliques. We know these almost-cliques in $\mathcal{K}_{\text{critical}}$ by Part (*6.*) of Section 4.5 (the holey ones are already colored and it is possible, yet unlikely, that even some of unholey ones are also colored in Phase 4). We color the remainder of $\mathcal{K}_{\text{critical}}$ now. In the previous phases, we solely colored vertices from lists $L(\cdot)$ sampled in `palette-sampling`. But we already know that such an approach is just not going to work for unholey critical almost-cliques (recall the example in Figure 1b discussed in Section 2.1). This is the first time we deviate from this approach (and thus deviate from palette sparsification-type arguments).

We now will use the critical-helper structures (Definition 4.6)—which our streaming algorithm collected in Part (*6.*) of Section 4.5—and a new "out of palette" coloring argument, wherein we color one of the vertices of the almost-clique using a color not sampled for it, so that *two* vertices of the almost-clique are colored the *same*. We then show that this already buys us enough flexibility to color the remaining vertices using lists $L_5(\cdot)$ of vertices similar to Phase 4. The coloring in this phase is also an extension of the last one.

— **Phase 6 – Unholey Friendly (or Social) Small Almost-Cliques (Section 5.6):** It can be verified, after a moment of thought or better yet by consulting Table 1, that the only almost-cliques remained to color are the ones that are unholey, small, and also not lonely. They are perhaps the "most problematic" ones and are handled last.[15] These almost-cliques also require the "out of palette" coloring argument used in Phase 5, but even this is not enough

for them (we already discussed this regarding Figure 5b in Section 2.2). In particular, unlike Phase 4 and 5 that allowed for coloring of the almost-cliques even in the presence of adversarial coloring of outside vertices, this simply cannot be true for this phase (as shown in Figure 5b); at the same time, unlike Phase 2 almost-cliques, we cannot hope for a "random" coloring of outside vertices.

To handle these almost-cliques, we rely on our friendly-helper structures (Definition 4.7) combined with a **recoloring step**: in particular, we recolor one vertex outside of the almost-clique using the sampled lists $L_6(\cdot)$ and show that this recoloring, plus another out of palette coloring argument, again buys us enough flexibility to color these almost-cliques also from lists $L_6(\cdot)$ (we note that this out of palette coloring argument is in fact different from the one used in Phase 5). Finally, due to the recoloring step, the coloring in this phase is no longer an extension of the last one.

After all these phases, we have finished coloring all the vertices. In other words, we now have Δ-coloring of the entire graph as desired. This will then conclude the proof of Theorem 1.1.

In the rest of this section, we go over each of these phases in details and present the algorithm and analysis for each one (postponing the less novel ones to Appendix B). We again emphasize that to find the final Δ-coloring, this phases must be executed *in this particular order*.

5.1 Phase 1: One-Shot Coloring

We start with the standard *one-shot coloring* algorithm used extensively in the coloring literature (to the best of our knowledge, this idea has appeared first in [51]). The purpose of this algorithm is to color many *pairs* of vertices in the neighborhood of sparse vertices using the *same* color (recall that neighborhood of sparse vertices contains many non-edges which can potentially be colored the same). This then effectively turn the sparse vertices into "low degree" ones and reduces the problem from Δ-coloring to $O(\Delta)$-coloring which is much simpler.

We have the following basic observation about the correctness of `one-shot-coloring`.

OBSERVATION 5.1. *The partial coloring C_1 computed by* `one-shot-coloring` *is a proper partial Δ-coloring in G.*

PROOF. It is immediate to verify that C_1 is a proper partial coloring in H simply because we remove both colors of any monochromatic edge. To see this also holds in G, note that for any edge $(u, v) \in G$, if $C_1(u) = C_1(v) \neq \perp$, then it means that $x(u) = x(v)$ which in particular also

15 It is quite natural to ask if these almost-cliques are the "hardest" to color, why do we wait to color them after everything else, at which time, our hands might be too tied? There are two closely related answers: (i) they may just be connected to each other (or rather the graph can only consists of these types of almost-cliques) and thus we anyway have to deal with at least one of them after having colored the rest of the graph; and (ii) even though they are "hard" to color, they are somewhat "more robust" also, compared to say Phase 2 almost-cliques, in that we can color them even when their outside neighbors are colored adversarially by using a key recoloring step.

Input: The vertex set V, the conflict-graph H, and the list $L_1(v)$ for every vertex $v \in V$.

(i) For every vertex $v \in V$:
- *Activate* v independently with probability $1/\alpha$ for parameter $\alpha = \Theta(1)$ in Eq. (1).
- If v is activated, set $x(v)$ to be the only color in $L_1(v)$, otherwise set $x(v) = \perp$.

(ii) For every vertex $v \in V$, set $C_1(v) = x(v)$ if for all $u \in N_H(v)$, $x(v) \neq x(u)$; otherwise, set $C_1(v) = \perp$. In words, any activated vertex v keeps its color $x(v)$ iff it is not used anywhere in its neighborhood.

Algorithm 4. The `one-shot-coloring` algorithm.

means $L_1(u) \cap L_1(v) \neq \emptyset$. Thus, the conflict graph H contains the edge (u, v) also, a contradiction with C_1 being a proper partial coloring of H. ■

We now get to the main property of `one-shot-coloring`. The effect of this partial coloring is that the neighborhood of every *sparse* vertex $v \in V$ becomes abundant with available colors (compared to the remaining degree of v). In particular, recall the definition of $\text{coldeg}_{C_1}(v)$ as the colored degree of a vertex v and $\text{avail}_{C_1}(v)$ as the number of colors available to v with respect to a partial coloring C_1 (defined in Section 3). Then we have the following guarantee for the `one-shot-coloring` algorithm:

LEMMA 5.2. *For every sparse vertex $v \in V_{\text{sparse}}$, in the partial coloring C_1 of* `one-shot-coloring`*,*

$$\text{avail}_{C_1}(v) > (\deg(v) - \text{coldeg}_{C_1}(v)) + \frac{\varepsilon^2 \cdot \Delta}{2\alpha}$$

with high probability, where the randomness is only over the choice of the lists $L_1(v)$.

The proof is postponed to Appendix B.1.

5.2 Phase 2: Lonely (or Social) Small Almost-Cliques

In this section, we will describe an algorithm that extends the partial coloring C_1 of Phase 1 to all small almost-cliques that are in $\mathcal{K}_{\text{lonely}}$ of Part (*5.*) of Section 4.5 (which in particular, contains all lonely almost-cliques and no friendly almost-clique). We are going to work with palette graphs introduced in Section 3.3 in this phase. The algorithm is simply as follows.

In Phase 2, we start by setting C to be equal to C_1 of Phase 1 and then successively run Algorithm 5 on each small almost-clique $K \in \mathcal{K}_{\text{lonely}}$ while updating C as described by the algorithm. At the end, we let C_2 denote the final partial Δ-coloring.

Input: A proper partial Δ-coloring C, a small almost-clique $K \in \mathcal{K}_{\text{lonely}}$, the conflict-graph H, and the list $L_2(v)$ for every vertex $v \in K$.

(i) Construct the sampled palette graph $\mathcal{G}_{\texttt{Sample}} = (\mathcal{L}, \mathcal{R}, \mathcal{E}_{\texttt{Sample}})$ of the almost-clique K, C, and $\mathcal{S} := \{L_2(v) \mid v \in K\}$ (according to Definition 3.9).

(ii) Find an $\mathcal{L}$-perfect matching $\mathcal{M}$ in $\mathcal{G}_{\texttt{Sample}}$ and output 'fail' if it does not exists. Otherwise, update $C(v) = \mathcal{M}(v)$ where $\mathcal{M}(v)$ denotes the color corresponding to the color-node matched to the vertex-node v by the matching $\mathcal{M}$.

Algorithm 5. The algorithm of Phase 2 for coloring each small almost-clique in $\mathcal{K}_{\text{lonely}}$.

LEMMA 5.3. *With high probability, C_2 computed by Phase 2 is a proper partial Δ-coloring in G that is an extension of C_1 and colors all small almost-cliques in $\mathcal{K}_{\text{lonely}}$.*

The randomness in this lemma is only over the randomness of `one-shot-coloring` *(activation probabilities) and choice of the lists $L_1(v)$ and $L_2(v)$ for all $v \in V$.*

We prove Lemma 5.3 in the rest of this subsection. The fact that C_2 is an extension of C_1 follows immediately from the definition of the algorithm as for every $K \in \mathcal{K}_{\text{lonely}}$, the corresponding $\mathcal{G}_{\texttt{Sample}}$ only contains vertices uncolored by C_1 and we never change the color of any colored vertex. Moreover, the fact that C_2 is a proper Δ-coloring follows from the definition of $\mathcal{G}_{\texttt{Sample}}$ as described in Section 3.3 (note that we only need edges in H and not all of G to construct $\mathcal{G}_{\texttt{Sample}}$).

The main part of the proof in this phase is to show that we actually succeed in coloring all small almost-cliques in $\mathcal{K}_{\text{lonely}}$ in this phase, i.e., w.h.p., Algorithm 5 does not ever return 'fail'.

Fix a small almost-clique $K \in \mathcal{K}_{\text{lonely}}$. Consider the base palette graph $\mathcal{G}_{\texttt{Base}} = (\mathcal{L}, \mathcal{R}, \mathcal{E}_{\texttt{Base}})$ of K and C (Definition 3.8) where C denotes the partial coloring passed to Algorithm 5 when coloring K. First, note that $|\mathcal{L}| \leqslant |\mathcal{R}|$ since $|K| \leqslant \Delta$ as K is small (Definition 4.1), and we remove at most one color-node in $\mathcal{R}$ per each vertex-node in $\mathcal{L}$ removed from K. Thus, having an $\mathcal{L}$-perfect matching in $\mathcal{G}_{\texttt{Base}}$ and $\mathcal{G}_{\texttt{Sample}}$ is not *entirely* out of the question. We now establish two other properties of $\mathcal{G}_{\texttt{Base}}$ that will allow us to argue that $\mathcal{G}_{\texttt{Base}}$ has an $\mathcal{L}$-perfect matching. We will then build on these properties to prove the same for $\mathcal{G}_{\texttt{Sample}}$ as well.

CLAIM 5.4. *Every vertex in $\mathcal{L}$ in $\mathcal{G}_{\texttt{Base}}$ has degree at least $3\Delta/4$, with high probability.*

Proof. Fix a vertex-node $v \in \mathcal{L}$ (and $v \in K$). By property iii). of Definition 3.4, v has at most $10\varepsilon\Delta$ neighbors outside K, each of which can rule out at most one color for v. As for neighbors inside K, recall that each vertex $u \in K$ activates in Algorithm 4 with probability $1/\alpha$. This

implies that in expectation, at most Δ/α of them can receive a color in C_1 (and hence C). By an application of Chernoff bound (Proposition 3.2), at most $2\Delta/\alpha < \Delta/100$ vertices of K are colored by C_1 with high probability (for the choice of α in Eq. (1)).

Hence, in total, only $10\varepsilon\Delta + \Delta/100$ colors are ruled out for v by C with high probability. Given the value of ε in Eq. (1), this means $\deg_{\mathcal{G}_{\text{Base}}}(v) > 3\Delta/4$ with high probability. ◆

The following claim—albeit simple to prove after having setup the process carefully—is the heart of the argument in this phase. Roughly speaking, this claim allows us to treat the coloring outside the almost-clique as "not too adversarial".

CLAIM 5.5. *Every vertex in $\mathcal{R}$ in $\mathcal{G}_{\text{Base}}$ has degree at least $3\Delta/4$, with high probability.*

PROOF. To lower bound the degree of a color-node $c \in \mathcal{R}$, we have to work a little harder. The main idea is this: for a color-node c to lose its edges to $\Omega(\Delta)$ vertex-nodes in $\mathcal{L}$, the color c itself must have been used to color $\Omega(\beta)$ vertices outside K by C; this is because of the crucial condition that K is *not* friendly and hence each vertex receiving the color c outside of K only rules c out for less than $2\Delta/\beta$ vertex-nodes of $\mathcal{L}$. It is generally very hard to keep track of which colors are assigned by C so far in the neighborhood of K, but fortunately we have a loose but simple proxy for that: as C is using only the colors in $L_1(\cdot)$ and $L_2(\cdot)$ at this point, we can simply consider which colors are sampled in these lists in the neighborhood of K instead. We formalize this in the following.

Fix a color-node $c \in \mathcal{R}$. For every vertex $v \notin K$, we define $m(v, K)$ as the number of edges from v to K in G. We define the random variable $X_{c,v}$ which is equal to $m(v, K)$ iff $c \in L_1(v) \cup L_2(v)$ and otherwise $X_{c,v} = 0$. Notice that $X_{c,v}$ is a (potentially loose) upper bound on the *reduction* in the degree of color-node $c \in \mathcal{R}$ because of any assignment of a color the vertices outside of K. In other words, we have that,

$$\deg_{\mathcal{G}_{\text{Base}}}(c) \geqslant |\mathcal{L}| - \sum_{v \notin K} X_{c,v}\,. \tag{3}$$

We use the variable $X_{c,v}$, instead of the actual color assignment of v by C, for two reasons: One, it is easy to compute its probability, and second (and more importantly) these variables for different v's are *independent* (while the actual color of vertices will be correlated). We would like to show that random variable $X_c := \sum_{u \notin K} X_{c,u}$ is sufficiently small.

Recall that $c \in L_1(v)$ with probability $1/\Delta$, and it is in $L_2(v)$ with probability β/Δ by the choice of lists in `palette-sampling` (Algorithm 1). Thus, it is in the union of the two lists with

probability at most $2\beta/\Delta$. Hence,

$$\begin{aligned}
\mathbb{E}[X_c] &\leqslant \sum_{v \notin K} m(v,K) \cdot \frac{2\beta}{\Delta} = \sum_{u \in K} |N(u) \setminus K| \cdot \frac{2\beta}{\Delta} \\
&\qquad \text{(by a simple double counting argument for edges between } K \text{ and its neighbors)} \\
&\leqslant |K| \cdot 10\varepsilon\Delta \cdot \frac{2\beta}{\Delta} \qquad \text{(by property } iii). \text{ of almost-cliques in Definition 3.4)} \\
&\leqslant \Delta \cdot 20\varepsilon \cdot \beta \qquad \text{(as } |K| \leqslant \Delta \text{ since } K \text{ is small by Definition 4.1)} \\
&\leqslant \Delta/100. \qquad \text{(by the choice of } \varepsilon, \beta \text{ in Eq. (1))}
\end{aligned}$$

To prove a concentration bound for X_c, note that it is a sum of independent random variables in the range $[0, 2\Delta/\beta]$, as each vertex $v \notin K$ has less than $2\Delta/\beta$ neighbors in K (as K is not a friendly almost-clique and thus has no friend neighbors—see Definition 4.4). Thus, by Chernoff bound (Proposition 3.2 for $b = 2\Delta/\beta$), we have,

$$\Pr\left(X_c > (1+20) \cdot \frac{\Delta}{100}\right) \leqslant \exp\left(-\frac{20^2 \cdot \Delta/100}{(3+20) \cdot 2\Delta/\beta}\right) = \exp\left(-\frac{400 \log n}{46}\right) < n^{-8},$$

by the choice of β in Eq. (1). A union bound over all choices of $v \notin K$ and $c \in [\Delta]$, combined with Eq. (3), implies that with high probability, $\deg_{\mathcal{G}_{\texttt{Base}}}(c) \geqslant |\mathcal{L}| - 21\Delta/100$.

Finally, as already proven in Claim 5.4, $|\mathcal{L}| \geqslant |K| - \Delta/100$ with high probability as at most $\Delta/100$ vertices of K are colored by C_1. Given that size of K is also at least $(1-5\varepsilon)\Delta$ by property i). of almost-cliques in Definition 3.4, and by the choice of ε in Eq. (1), we get that with high probability $|\mathcal{L}| \geqslant \Delta - 2\Delta/100$. Combined with the above bound, we have

$$\deg_{\mathcal{G}_{\texttt{Base}}}(c) \geqslant \Delta - 2\Delta/100 - 21\Delta/100 > 3\Delta/4$$

as desired, concluding the proof. ∎

Given that size of $\mathcal{L}$ in $\mathcal{G}_{\texttt{Base}}$ is at most Δ, it is now easy to use Claims 5.4 and 5.5, combined with Hall's theorem (Fact 3.1) to prove that $\mathcal{G}_{\texttt{Base}}$ has an $\mathcal{L}$-perfect matching. But, our goal is to prove that $\mathcal{G}_{\texttt{Sample}}$, not only $\mathcal{G}_{\texttt{Base}}$, has such a matching; this is the content of the next claim.

CLAIM 5.6. *The subgraph $\mathcal{G}_{\texttt{Sample}}$ has an $\mathcal{L}$-perfect matching with high probability.*

PROOF. We condition on the high probability events that $\mathcal{G}_{\texttt{Base}}$ has the properties in Claims 5.4 and 5.5. An important observation is in order: the properties of $\mathcal{G}_{\texttt{Base}}$ depend on the choice of $L_1(v)$ for $v \in V$ but only $L_2(v)$ for $v \notin K$ (as we only need to visit the coloring of C with $L_2(\cdot)$ *outside* of K). As a result, even conditioned on these properties, the choice of $L_2(v)$ for $v \in K$ is independent and from its original distribution in `palette-sampling`.

At this point, $\mathcal{G}_{\texttt{Sample}}$ is a subgraph of $\mathcal{G}_{\texttt{Base}}$ obtained by sampling each edge independently and with probability β/Δ by the choice of $L_2(\cdot)$ in `palette-sampling` (Algorithm 1). We use this to prove that $\mathcal{G}_{\texttt{Sample}}$ should also have an $\mathcal{L}$-perfect matching with high probability. The

argument follows standard ideas in random graph theory (even though $\mathcal{G}_{\texttt{Sample}}$ is not exactly a random graph).

By Hall's theorem (Fact 3.1), for $\mathcal{G}_{\texttt{Sample}}$ to *not* have an $\mathcal{L}$-perfect matching, there should exist a set $A \subseteq \mathcal{L}$ such that $\left|N_{\mathcal{G}_{\texttt{Sample}}}(A)\right| < |A|$. But for this to happen, there should exist a pair (S,T) of subsets of $\mathcal{L}$ and $\mathcal{R}$, respectively, such that $|T| = |S| - 1$ and no edge between S and $\mathcal{R} \setminus T$ is sampled in $\mathcal{G}_{\texttt{Sample}}$ (simply take $A = S$ and notice that $N(A) \subseteq T$ which has size less than A). We refer to any such pair (S,T) as a **witness pair**. We bound the probability that any witness pair exists in $\mathcal{G}_{\texttt{Sample}}$.

Case 1: when $|S| \leqslant 2\Delta/3$. Consider any choice of the set T with $|T| = |S|-1$ from $\mathcal{R}$. By Claim 5.4, degree of every vertex-node in S is at least $3\Delta/4$ in $\mathcal{G}_{\texttt{Base}}$. This means the number of edges from S to $\mathcal{R} \setminus T$ is at least $|S| \cdot (3\Delta/4 - 2\Delta/3) = |S| \cdot \Delta/12$. As such,

$$\Pr\left((S,T) \text{ is a witness pair}\right) \leqslant \left(1 - \frac{\beta}{\Delta}\right)^{|S|\cdot\Delta/12} \leqslant \exp\left(-\frac{100}{12} \cdot |S| \cdot \log n\right) < n^{-8|S|},$$

by the choice of β in Eq. (1). A union bound over all $\binom{|\mathcal{R}|}{|S|-1} < n^{|S|-1}$ choices for T then implies that for any such S,

$$\Pr\left(\text{there is a set } T \text{ so that } (S,T) \text{ is a witness pair}\right) \leqslant n^{|S|-1} \cdot n^{-8|S|} < n^{-7|S|}.$$

Finally, a union bound all choices for the set S, partitioned based on their size, implies that,

$$\Pr\left(\text{there is a witness pair } (S,T) \text{ with } |S| \leqslant 2\Delta/3\right) \leqslant \sum_{s=1}^{2\Delta/3} \binom{|\mathcal{L}|}{s} \cdot n^{-7|s|} < n^{-5}.$$

Case 2: when $|S| > 2\Delta/3$. Again, fix any choice of the set T with $|T| = |S| - 1$ from $\mathcal{R}$. This time, by Claim 5.5, degree of every color-node *not* in T is at least $3\Delta/4$ in $\mathcal{G}_{\texttt{Base}}$. This means that neighborhood of each such color-node intersects with S in at least $3\Delta/4 - (\Delta - 2\Delta/3) = 5\Delta/12$ (as $|\mathcal{L}| \leqslant \Delta$) vertex-nodes. In other words, there are at least $|\mathcal{R} \setminus T| \cdot 5\Delta/12$ edges between S and $\mathcal{R} \setminus T$ in $\mathcal{G}_{\texttt{Base}}$. Thus,

$$\Pr\left((S,T) \text{ is a witness pair}\right) \leqslant \left(1 - \frac{\beta}{\Delta}\right)^{|\mathcal{R}\setminus T|\cdot 5\Delta/12} \leqslant \exp\left(-\frac{500}{12} \cdot |\mathcal{R} \setminus T| \cdot \log n\right) < n^{-40|\mathcal{R}\setminus T|},$$

by the choice of β in Eq. (1). A union bound over all $\binom{|\mathcal{R}|}{|T|} = \binom{|\mathcal{R}|}{|\mathcal{R}\setminus T|} < n^{|\mathcal{R}\setminus T|}$ choices for T then implies that for any such S,

$$\Pr\left(\text{there is a set } T \text{ so that } (S,T) \text{ is a witness pair}\right) \leqslant n^{|\mathcal{R}\setminus T|} \cdot n^{-40|\mathcal{R}\setminus T|} \leqslant n^{-39\cdot(|\mathcal{R}|-|S|+1)},$$

where we used the fact that $|T| = |S| - 1$. Now note that the number of choices for the set S of a fixed size is

$$\binom{|\mathcal{L}|}{|S|} = \binom{|\mathcal{L}|}{|\mathcal{L}| - |S|} \leqslant n^{|\mathcal{L}|-|S|} \leqslant n^{|\mathcal{R}|-|S|},$$

where the last inequality uses the fact that $|\mathcal{R}| \geqslant |\mathcal{L}|$. As a result,

$$\Pr\left(\text{there is a witness pair } (S,T) \text{ with } |S| > 2\Delta/3\right) \leqslant \sum_{s=2\Delta/3}^{|\mathcal{L}|} n^{|\mathcal{R}|-|S|} \cdot n^{-39\cdot(|\mathcal{R}|-|S|+1)} < n^{-39}.$$

Finally, by combining Case 1 and 2 above, we have that with high probability, there is no witness set (S,T) in $\mathcal{G}_{\texttt{Sample}}$. This implies that for every $A \subseteq \mathcal{L}$, we have $\left|N_{\mathcal{G}_{\texttt{Sample}}}(A)\right| \geqslant |A|$, which, by Hall's theorem (Fact 3.1), implies that $\mathcal{G}_{\texttt{Sample}}$ has an $\mathcal{L}$-perfect matching. ■

Lemma 5.3 now follows immediately from Claims 5.4 to 5.6 as described earlier.

REMARK 5.7. Before moving on from this subsection, let us mention why our coloring procedure attempts to color lonely small almost-cliques (Phase 2) before the remaining sparse vertices. In Claim 5.5, we crucially used the fact that we can use the lists used to color C so far as a proxy for approximating the event $C(v) = c$ with $c \in L_1(v) \cup L_2(v)$, instead. This was okay because these lists are of relatively small size to make the argument go through. However, coloring sparse vertices requires us to use the lists $L_3(\cdot)$ which are much larger and thus would break this claim entirely.

Concretely, in Claim 5.5, we had $O(\varepsilon\Delta^2)$ edges going out of the almost-clique and each was responsible for blocking a fixed color on a vertex with probability $p = O(\beta/\Delta)$ which is governed by sizes of $L_1(\cdot), L_2(\cdot)$. This meant that each color was blocked for $O(\varepsilon \cdot \beta \cdot \Delta)$ vertices which can be made $o(\Delta)$ by taking ε sufficiently smaller than β. Nevertheless, had we also included lists $L_3(\cdot)$, then the right probability parameter p would have become $O(\beta/(\varepsilon^2\Delta))$ which is crucial for coloring ε-sparse vertices; but then, it meant that the bound we got on the number of blocked vertices for a color is actually $O(\varepsilon\Delta^2 \cdot \beta/(\varepsilon^2\Delta)) = O(\Delta/\varepsilon)$ which is $> \Delta$ no matter the tuning of parameters.

5.3 Phase 3: Sparse Vertices

In this phase, we will describe an algorithm to extend the partial coloring C_2 to all vertices of V_{sparse}. The key observation is that for any extension of C_1 (i.e. C_2, and every intermediate coloring in this phase), the gap between available colors and remaining degree created by C_1 for each sparse vertex (see Lemma 5.2) does not shrink. This is because if C is an extension of C_1, each additional neighbor of some sparse vertex v that C colors, increases $\mathsf{coldeg}_C(v)$ by 1, and decreases $\mathsf{avail}_C(v)$ by at most 1, keeping the gap intact. We have the following lemma for this phase:

LEMMA 5.8. *With high probability, there exists a proper partial Δ-coloring C that is an extension of C_2 and colors all remaining vertices in V_{sparse} using only the colors in the lists $L_3(v)$ for sparse vertices $v \in V_{\text{sparse}}$ (and thus the randomness is also only over these lists).*

Once again, since the lemma is not new, its proof is postponed to Appendix B.2.

We note that C is *not* the final coloring we obtain in this phase. Instead, we are going to update C to a proper Δ-coloring C_3 that colors all vertices in V_{sparse} as well as all small almost-cliques in $\mathcal{K}_{\text{lonely}}$ handled by Phase 2; however, we shall remove the color of every other vertex v, i.e., set $C_3(v) = \perp$ for them. Such vertices are solely colored by the `one-shot-coloring` algorithm and we no longer need their guarantees as we are done coloring sparse vertices. Thus, to summarize:

— C_3 is a proper partial Δ-coloring of all vertices in V_{sparse} as well as small almost-cliques in $\mathcal{K}_{\text{lonely}}$ and does not color any other vertex (we also require no further properties from C_3 and it might as well be considered adversarially chosen from now on).

5.4 Phase 4: Holey Almost-Cliques

In this phase, we will extend the partial coloring C_3 to all vertices in holey almost-cliques.

LEMMA 5.9. *There exists a proper partial Δ-coloring C_4 that is an extension of C_3 and assigns a color to every vertex v in each holey almost-clique using a color from $L_4(v)$. The randomness in this lemma is only over the lists $L_4(\cdot)$ of all vertices.*

As discussed in the overview of the coloring algorithm, we will iterate over all almost-cliques in our decomposition, and attempt to color them assuming that they are holey. The main tool to show that this succeeds on holey almost-cliques is the following lemma, which shows that any coloring outside a holey almost-clique can be extended to it while using only the lists $L_4(\cdot)$ on vertices inside it.

LEMMA 5.10. *For a holey almost-clique K, and any partial Δ-coloring C outside K, there exists, with high probability, a coloring C' which extends C to K such that $C'(v) \in L_4(v)$ for all $v \in K$.*

The proof is almost verbatim from [7], except that we have to go through every step carefully to make sure it works for Δ-coloring—hence we provide it in Appendix B.3 for completeness.

Lemma 5.9 now follows immediately from Lemma 5.10 by going over all uncolored almost-cliques at this point one by one, and apply this lemma with C being the current coloring, and C' being the one we can update this coloring to. Thus, at the end, we obtain the desired C_4.

5.5 Phase 5: Unholey Critical Almost-Cliques

In this phase, we will color unholey critical almost-cliques. In particular, we have a set $\mathcal{K}_{\text{critical}}$ of almost-cliques, and for each $K \in \mathcal{K}_{\text{critical}}$, a critical-helper structure $(u, v, N(v))$ (Definition 4.6). The main lemma of this section is:

LEMMA 5.11. *With high probability, there exists a proper partial Δ-coloring C_5 that is an extension of C_4 and satisfies the following properties: (i) it colors vertices of all almost-cliques in*

$\mathcal{K}_{\text{critical}}$; and (ii) for any vertex v, if $C_5(v) \notin L(v)$, then $N_{H^+}(v) = N_G(v)$; that is, we can only color v with a color not from $L(v)$ if we know its entire neighborhood (via the critical helper structure).

The randomness in this lemma is only over the lists $L_5(v)$ for vertices v in $\mathcal{K}_{\text{critical}}$.

This is the first phase in which we use our out-of-palette-coloring idea – we do not require that $C_5(v) \in L(v)$ always holds in this lemma. In particular, for the vertex v in the critical-helper $(u, v, N(v))$, we are going to use a color out of its sampled palette. Since we know the entire neighborhood of v, we at least have enough information to avoid an improper coloring.

The proof of the lemma is algorithmic. We start with a brief overview. The plan (as always) is to iterate over the almost-cliques of $\mathcal{K}_{\text{critical}}$ in arbitrary order, and extend the coloring C_4 to eventually color all of them. For a particular almost-clique $K \in \mathcal{K}_{\text{critical}}$, we will use the critical-helper structure $(u, v, N(v))$ to assign the *same* color to both u and v. The reason we can do this is that we know all the neighbors of v, and hence can pick a color in the list $L_5(u)$ that can be assigned to both of them – existence of such a color in the first place is because both u and v belong to an almost-clique and thus have at most $O(\varepsilon\Delta)$ edges to outside; thus, as long as we have sampled a color out of these many, which will happen with high probability, we can find such a color. Having done that, the rest of K can be colored by palette sparsification: the imbalance we create by giving two vertices the same color is just enough for it to succeed with high probability.

Input: critical almost-cliques $\mathcal{K}_{\text{critical}}$, a critical-helper structure for each $K \in \mathcal{K}_{\text{critical}}$, and the partial coloring C_4.

(i) Initialize $C \leftarrow C_4$. For each $K \in \mathcal{K}$:

- Let $(u, v, N(v))$ be the critical-helper structure for K of Part (6.) of Section 4.5.
- Find a color $c \in L_5(u)$ that is not used in $N_{H^+}(u) \cup N_{H^+}(v)$ by C, and set

$$C(u) \leftarrow c \quad \text{and} \quad C(v) \leftarrow c.$$

- Extend C to color $K \setminus \{u, v\}$ with the color of each vertex w chosen from $L_5(w)$, by constructing the sampled palette graph of K with respect to C and sampled lists $\mathcal{S} := \{L_5(w) \mid w \in K\}$ exactly as in Algorithm 5 (see Claim 5.13 for details).

(ii) Return $C_5 \leftarrow C$ as the output coloring.

Algorithm 6. The `unholey-critical-coloring` algorithm.

This algorithm claims to color all of the almost-cliques in $\mathcal{K}_{\text{critical}}$, but it is not obvious at all that each of its steps is possible. We will prove this in the following series of claims.

CLAIM 5.12. *With high probability, there exists a color $c \in L_5(u)$ such that c is not used by C in $N_G(u) \cup N_G(v)$. Further, there is an algorithm that can find this color if it exists.*

PROOF. Recall that u and v have at most $10\varepsilon\Delta$ neighbors each outside K, and hence only at most $20\varepsilon\Delta$ neighbors in total which are colored by C (see Sidefigure 9).

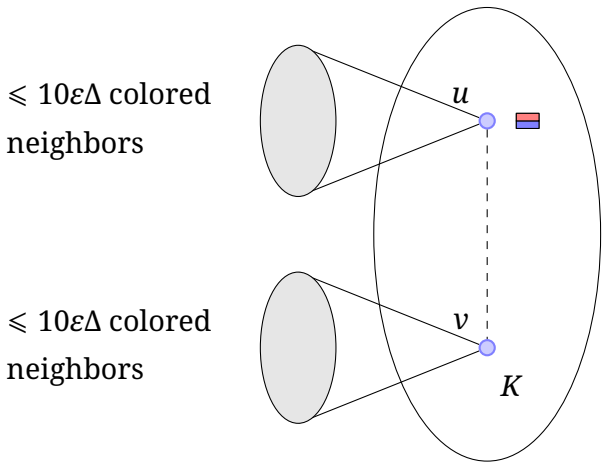


Figure 9. $L_5(u)$ has a color not used by C over $N(u) \cup N(v)$.

Let $\mathsf{Avail}_C(u, v)$ be the set of colors in $[\Delta]$ that are not used in the neighborhood of u or v by the coloring C. Then

$$|\mathsf{Avail}_C(u, v)| \geqslant \Delta - 20\varepsilon\Delta \geqslant \Delta/2.$$

We would like to show that $\Pr(L_5(u) \cap \mathsf{Avail}_C(u, v) = \emptyset)$ is small. Since each color from $\mathsf{Avail}_C(u, v)$ is sampled into $L_5(u)$ independently with probability β/Δ, the probability that none of them are in $L_5(u)$ is at most:

$$\left(1 - \frac{\beta}{\Delta}\right)^{\Delta/2} \leqslant \exp\left(-\frac{\beta \cdot \Delta}{2\Delta}\right) = n^{-50}.$$

Hence a "good" color exists with high probability.

To find this color, we note that the critical-helper structure contains $N(v)$ and while we do not necessarily know all the neighbors of u in G, we do know the ones that:

— Have a color from $L_5(u)$ in their own palette.
— Were assigned a color *outside* their palette – since for such vertices we know all of their neighbors by the invariant maintained in Lemma 5.11.

This means that we can iterate over the colors in $[\Delta]$, and check for each one whether it is used by $N_{H^+}(u) \cup N_{H^+}(v)$, which serves as a proxy for checking $N_G(u) \cup N_G(v)$, and we are done. ■

We now have to perform a somewhat daunting task—to extend the coloring C to the rest of the almost-clique K. It turns out that the small imbalance we create in the previous line (by coloring 2 vertices with 1 color) is enough for palette sparsification to come to our rescue.

CLAIM 5.13. *With high probability, $K \setminus \{u, v\}$ can be colored as an extension of C, while coloring each vertex $w \in K \setminus \{u, v\}$ with a color from $L_5(w)$.*

PROOF. Consider the base and the sampled palette graphs $\mathcal{G}_{\texttt{Base}}$ and $\mathcal{G}_{\texttt{Sample}}$ (Definitions 3.8 and 3.9) corresponding to the partial coloring C, with:

— $\mathcal{L} := K \setminus \{u, w\}$,
— $\mathcal{R} := [\Delta] \setminus \{c\}$, where c is the color assigned to u and v, and
— $S(v) := L_5(v)$ for all $v \in \mathcal{L}$.

Recall that an $\mathcal{L}$-perfect matching in $\mathcal{G}_{\texttt{Sample}}$ implies a coloring of remaining vertices in K from their $L_5(\cdot)$ lists which is further an extension of C. We are going to use Lemma 3.10 to obtain that $\mathcal{G}_{\texttt{Sample}}$ has a perfect matching.

For this, we need to establish the required properties of $\mathcal{G}_{\texttt{Base}}$. Proceeding in the same order as the lemma (recall that m denotes $|\mathcal{L}|$ in this lemma):

(i) $m \leqslant |\mathcal{R}| \leqslant 2m$: In this instance, $|L| = |R|$, so both inequalities follow trivially.

(ii) The minimum degree of any vertex in $\mathcal{L}$ is at least $2m/3$: Note that any vertex $v \in \mathcal{L}$ is in K, and hence has only at most $10\varepsilon\Delta$ edges going out of K (in G, the input graph). Hence there are at most $10\varepsilon\Delta$ colors that are blocked for v, and $\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |\mathcal{R}| - 10\varepsilon\Delta \geqslant (2/3) \cdot m$.

(iii) For any subset $A \subseteq \mathcal{L}$ such that $|A| \geqslant m/2$,

$$\sum_{v \in A} \deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |A| \cdot m - m/4.$$

Let t denote the number of non-edges in K, and recall that since K is unholey, $t \leqslant 10^7 \cdot \varepsilon\Delta < \Delta/10$. Note that any vertex $v \in K$ has at most $\Delta - (|K| - 1 - \overline{\deg}_K(v))$ neighbors (in G) outside K, because $\deg_G(v) \leqslant \Delta$, and v has exactly $|K| - 1 - \overline{\deg}_K(v)$ neighbors inside K. But since each non-edge between $v \in \mathcal{L}$ and a color in $\mathcal{R}$ corresponds to an edge from v to outside K, we have:

$$\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |R| - (\Delta + 1) + |K| - \overline{\deg}_K(v).$$

By summing this inequality for all $v \in A$, we get:

$$\begin{aligned}
\sum_{v \in A} \deg_{\mathcal{G}_{\texttt{Base}}}(v) &\geqslant \sum_{v \in A} |R| - (\Delta + 1) + |K| - \overline{\deg}_K(v) \\
&= \sum_{v \in A} \underbrace{\Delta - 1}_{|R|} - (\Delta + 1) + \underbrace{|\mathcal{L}| + 2}_{|K|} - \overline{\deg}_K(v) \\
&= \sum_{v \in A} |\mathcal{L}| - \overline{\deg}_K(v) = |A| \cdot m - \underbrace{\sum_{v \in A} \overline{\deg}_K(v)}_{\leqslant 2t} \\
&\geqslant |A| \cdot m - m/4.
\end{aligned}$$

Each edge of $\mathcal{G}_{\texttt{Base}}$ is sampled into $\mathcal{G}_{\texttt{Sample}}$ independently with probability

$$\beta/\Delta \geqslant 99 \log n/m \geqslant 20/m \cdot (\log n + 3 \cdot \log n).$$

By Lemma 3.10 $\mathcal{G}_{\texttt{Sample}}$ has a perfect matching with probability at least $(1 - n^{-3})$. ■

This concludes the proof of Lemma 5.11.

5.6 Phase 6: Unholey Friendly (or Social) Small Almost-Cliques

In this (final!) phase, we describe an algorithm that takes the partial coloring C_5 and from it compute a proper Δ-coloring of the entire graph by coloring the remaining vertices. This step however is the one that includes our *recoloring* ideas and thus the coloring we obtain is no longer an extension of C_5.

We are left with a set $\mathcal{K}_{\text{friendly}}$ of unholey small almost-cliques, that may be friendly or social. For each $K \in \mathcal{K}_{\text{friendly}}$, we have a friendly-helper $(u, v, w, N(v), N(w))$ (Definition 4.7). Note in particular that we will actually edit the color assigned to some vertices, so the coloring obtained by this algorithm is not necessarily an extension of C_5. We show the following lemma:

LEMMA 5.14. *Given the coloring C_5 from the previous section, with high probability there exists a proper coloring C_6 that colors the entire graph such that: For any vertex v, if $C_6(v) \notin L(v)$, $N_G(v) = N_{H^+}(v)$. That is, we can color v with a color not from $L(v)$ only if we know its entire neighborhood (via the friendly-helper structure).*

The randomness in this lemma is over the lists $L_{6,i}(v)$ for <u>all</u> vertices $v \in V$ and all $i \in [2\beta]$ (even including vertices that were colored before this phase).

Once again, we will prove the lemma with an algorithm. The idea is the same, to extend the coloring C_5 to each almost-clique in $\mathcal{K}_{\text{friendly}}$ one-by-one, with a key difference being that we will go back and edit the color of one vertex per almost-clique we color.

For an almost-clique K, we will use its friendly-helper structure $(u, v, w, N(v), N(w))$ to assign the same color to u and w, and then invoke palette sparsification to color the rest of K.

As before, while the algorithm claims to color all the almost-cliques in $\mathcal{K}_{\text{friendly}}$, it is far from obvious that each step it performs is possible. We show that this is indeed the case.

CLAIM 5.15. *With high probability, there exists a color in $L_{6,i_u}(u)$ that does not appear in C over $N(u) \cup N(w)$. Further, there is an algorithm that can find this color if it exists.*

PROOF. The crucial observation is that since u is not a stranger to K, it has at least Δ/β uncolored neighbors in the coloring C. Further, since $w \in K$ – an almost-clique – it has at most $10\varepsilon\Delta$ neighbors outside K, and hence only at most $10\varepsilon\Delta$ neighbors that receive colors in C. Hence there are at least $\Delta/\beta - 10\varepsilon\Delta \geqslant \Delta/2\beta$ colors that are not used by C in $N(u) \cup N(w)$ (see Sidefigure 10).

Input: The set of almost-cliques $\mathcal{K}_{\text{friendly}}$, a friendly-helper structure for each $K \in \mathcal{K}_{\text{friendly}}$, and the partial coloring C_5.

(i) Initialize $C = C_5$, and an index $i_u \leftarrow 0$ for each vertex $u \in V(G)$ (this keeps track of how many times we recolored u).

(ii) For each $K \in \mathcal{K}_{\text{friendly}}$:

— Let $(u, v, w, N(v), N(w))$ be the friendly-helper structure of K.

— Find a color $c \in L_{6,i_u}(u)$ such that $c \notin C(N_{H^+}(u) \cup N_{H^+}(w))$ and set

$$C(u) \leftarrow c \quad \text{and} \quad C(w) \leftarrow c.$$

Update $i_u \leftarrow i_u + 1$ (we emphasize that u is not part of K but rather a neighbor to it).

— Extend C to color $K \setminus \{v, w\}$ with each remaining vertex x getting a color from $L_{6,2\beta}(x)$ exactly as in Algorithm 6 (see Claim 5.16 for details).

— Extend C to color v by finding a color that does not appear in $N_{H+}(v)$.

(iii) Return $C_6 \leftarrow C$ as the output.

Algorithm 7. The `unholey-friendly-coloring` algorithm.

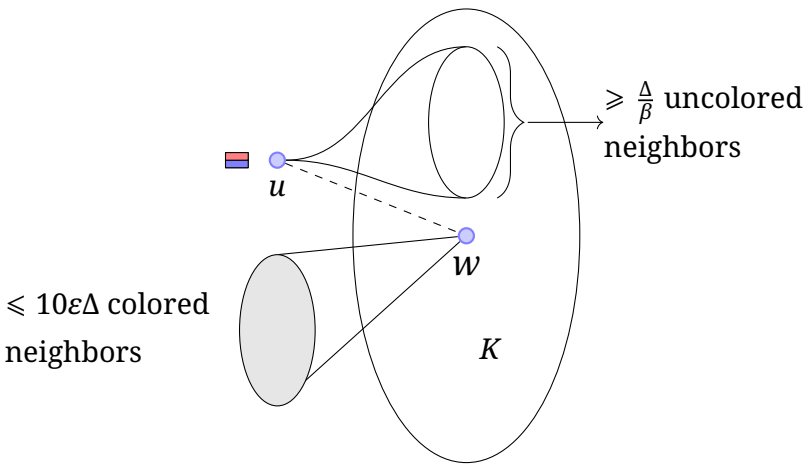


Figure 10. $L_{6,i_u}(u)$ has a color not used by C over $N(u) \cup N(w)$.

Let $\text{Avail}_C(u, w)$ denote this set of colors. We are interested in showing that the probability $\Pr\left(L_{6,i_u}(u) \cap \text{Avail}_C(u, w) = \emptyset\right)$ is small. We start with the technical note that this is the first time (any part of) the coloring algorithm is looking at $L_{6,i_u}(u)$, and hence this particular list is independent of C entirely. Recall that each color from $\text{Avail}_C(u, w)$ is in $L_{6,i_u}(u)$ independently with probability β^2/Δ. Hence the probability that none of them is in $L_{6,i_u}(u)$ is at most:

$$\left(1 - \frac{\beta^2}{\Delta}\right)^{\Delta/2\beta} \leqslant \exp\left(-\frac{\beta^2 \cdot \Delta}{2\beta \cdot \Delta}\right) = n^{-50}.$$

So there is a color c that satisfies our requirements with high probability. To find c we can iterate over the colors of $L_{6,i_u}(u)$ and check whether c is used by C in $N_{H^+}(u) \cup N_{H^+}(w)$. As before, this is a proxy for checking that c is used in $N_G(u) \cup N_G(w)$, and it works because $c \in L(u)$, $N_{H^+}(w) = N_G(w)$, and if any neighbor of u picked a color out of its palette, it will be in $N_{H^+}(u)$.

Note that we increment i_v in the final line to get a "fresh" list for the next time we have to color v—this happens at most β times for a vertex v (since it must be a non-stranger to some almost clique K to be recolored), and L_6 has 2β lists for each v, so there are always enough lists to go around. ■

Next, we have the palette sparsification analogue of this phase.

CLAIM 5.16. *With high probability, $K \setminus \{v, w\}$ can be colored as an extension of C, where the color of each vertex $x \in K \setminus \{v, w\}$ is from $L_{6,2\beta}(x)$.*

PROOF. Consider the base and the sampled palette graphs $\mathcal{G}_{\texttt{Base}}$ and $\mathcal{G}_{\texttt{Sample}}$ (Definitions 3.8 and 3.9) corresponding to the partial coloring C, with:

- $\mathcal{L} = K \setminus \{v, w\}$,
- $\mathcal{R} = [\Delta] \setminus \{c\}$, where c is the color assigned to u and w, and
- $S(x) = L_{6,2\beta}(x)$ for all $x \in \mathcal{L}$.

Recall that an $\mathcal{L}$-perfect matching in $\mathcal{G}_{\texttt{Sample}}$ gives a coloring which extends C to $\mathcal{L}$ using only colors from the list $L_{6,2\beta}$. Hence our focus shifts to showing that $\mathcal{G}_{\texttt{Base}}$ has the properties required by Lemma 3.10 to obtain that $\mathcal{G}_{\texttt{Sample}}$ has a perfect matching. Proceeding in the same order as the lemma:

(i) $m \leqslant |\mathcal{R}| \leqslant 2m$: The first inequality follows from the fact that $|\mathcal{R}| = \Delta - 1$, and $m \leqslant \Delta - 2$ (since K is a *small* almost-clique, and we removed two vertices from it). For the second one, note that $2m = 2|K| - 4 \geqslant 3/2 \cdot \Delta$.

(ii) The minimum degree of any vertex in $\mathcal{L}$ is at least $2m/3$. Note that any vertex $v \in \mathcal{L}$ is in K, and hence has only at most $10\varepsilon\Delta$ edges going out of K (in G, the input graph). Hence there are at most $10\varepsilon\Delta$ colors that are blocked for v, and $\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |\mathcal{R}| - 10\varepsilon\Delta \geqslant 2m/3$ (the last inequality is from combining parts (i) and (ii) above).

(iii) For any subset $S \subset \mathcal{L}$ such that $|S| \geqslant m/2$,

$$\sum_{v \in S} \deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant (|S| \cdot m) - m/4.$$

Since the only facts we use are that K is unholey, and points (i) and (ii) above, the proof is exactly the same as in Claim 5.13.

And once again, by invoking Lemma 3.10 we are done. ■

To finish up, note that v has two neighbors (u and w) with the same color, and we know its entire neighborhood. Hence we can find a color to assign to it, and we are done.

This concludes the proof of Lemma 5.14. As at this point, all vertices of the graph are colored, we obtain a proper Δ-coloring of G. This in turn concludes the proof of Theorem 1.1.

Acknowledgements

Sepehr Assadi would like to thank Soheil Behnezhad, Amit Chakrabarti, Prantar Ghosh, and Merav Parter for helpful conversations. We thank the organizers of DIMACS REU in Summer 2020, in particular Lazaros Gallos, for initiating this collaboration and all their help and encouragements along the way. We are also thankful to the anonymous reviewers of STOC 2022 and TheoretiCS for helpful comments and suggestions on the presentation of the paper.

References

[1] Lecture notes on palette sparsification for $(\Delta+1)$ coloring by Sepehr Assadi. https://sepehr.assadi.info/courses/cs671-f20/lec3.pdf, 2020. Accessed: 2023-05-08 (17).

[2] **Amir Abboud**, **Keren Censor-Hillel**, **Seri Khoury**, **and Ami Paz**. Smaller cuts, higher lower bounds. *ACM Trans. Algorithms*, 17(4):30:1–30:40, 2021 DOI (5).

[3] **Kook Jin Ahn**, **Sudipto Guha**, **and Andrew McGregor**. Analyzing graph structure via linear measurements. *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012 DOI (5, 13).

[4] **Noga Alon and Sepehr Assadi**. Palette sparsification beyond (Δ+1) vertex coloring. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, 6:1–6:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020 DOI (2–6, 8).

[5] **Noga Alon and Michael Tarsi**. Colorings and orientations of graphs. *Comb.* 12(2):125–134, 1992 DOI (3).

[6] **Sepehr Assadi**, **Andrew Chen**, **and Glenn Sun**. Deterministic graph coloring in the streaming model. *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 261–274. ACM, 2022 DOI (2, 4, 6).

[7] **Sepehr Assadi**, **Yu Chen**, **and Sanjeev Khanna**. Sublinear algorithms for (Δ + 1) vertex coloring. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019 DOI (2–10, 12, 15–19, 23–25, 35, 36, 44).

[8] **Sepehr Assadi**, **Pankaj Kumar**, **and Parth Mittal**. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 234–247. ACM, 2022 DOI (1, 6).

[9] **Sepehr Assadi and Chen Wang**. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022 DOI (12, 15, 16).

[10] **Bradley Baetz and David R. Wood**. Brooks' vertex-colouring theorem in linear time. *CoRR*, abs/1401.8023, 2014 (3, 6).

[11] **Alkida Balliu**, **Sebastian Brandt**, **Fabian Kuhn**, **and Dennis Olivetti**. Distributed Δ-coloring plays hide-and-seek. *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 464–477. ACM, 2022 DOI (6).

[12] **Paul Beame**, **Paraschos Koutris**, **and Dan Suciu**. Communication steps for parallel query processing. *J. ACM*, 64(6):40:1–40:58, 2017 DOI (5).

[13] **Soheil Behnezhad**, **Mahsa Derakhshan**, **MohammadTaghi Hajiaghayi**, **Marina Knittel**, **and Hamed Saleh**. Streaming and massively parallel algorithms for edge coloring. *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019 DOI (2, 6).

[14] **Suman K. Bera**, **Amit Chakrabarti**, and **Prantar Ghosh**. Graph coloring via degeneracy in streaming and other space-conscious models. *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, 11:1–11:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020 DOI (2–4, 6, 8).

[15] **Suman Kalyan Bera and Prantar Ghosh**. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018 (2, 6, 8).

[16] **Anup Bhattacharya**, **Arijit Bishnu**, **Gopinath Mishra**, and **Anannya Upasana**. Even the easiest(?) graph coloring problem is not easy in streaming! *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021 DOI (2, 6).

[17] **Oleg V. Borodin and Alexandr V. Kostochka**. On an upper bound of a graph's chromatic number, depending on the graph's degree and density. *J. Comb. Theory, Ser. B*, 23(2-3):247–250, 1977 DOI (6).

[18] **Sebastian Brandt**, **Orr Fischer**, **Juho Hirvonen**, **Barbara Keller**, **Tuomo Lempiäinen**, **Joel Rybicki**, **Jukka Suomela**, and **Jara Uitto**. A lower bound for the distributed Lovász local lemma. *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 479–488. ACM, 2016 DOI (3, 6).

[19] **Rowland Leonard Brooks**. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37 of number 2, pages 194–197. Cambridge University Press, 1941 (3).

[20] **Harry Buhrman and Ronald de Wolf**. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.* 288(1):21–43, 2002 DOI (7, 54).

[21] **Mark Bun and Justin Thaler**. Dual lower bounds for approximate degree and markov-bernstein inequalities. *Inf. Comput.* 243:2–25, 2015 DOI (55).

[22] **Amit Chakrabarti**, **Prantar Ghosh**, and **Manuel Stoeckl**. Adversarially robust coloring for graph streams. *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, 37:1–37:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022 DOI (2, 6).

[23] **Yi-Jun Chang**, **Wenzheng Li**, and **Seth Pettie**. An optimal distributed $(\Delta+1)$-coloring algorithm? *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 445–456. ACM, 2018 DOI (15).

[24] **Graham Cormode**, **Jacques Dark**, and **Christian Konrad**. Independent sets in vertex-arrival streams. *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019 DOI (5).

[25] **Daniel W. Cranston and Landon Rabern**. Brooks' theorem and beyond. *J. Graph Theory*, 80(3):199–225, 2015 DOI (3).

[26] **Artur Czumaj**, **Peter Davies**, and **Merav Parter**. Improved deterministic $(\Delta+1)$ coloring in low-space MPC. *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 469–479. ACM, 2021 DOI (6).

[27] **Artur Czumaj**, **Peter Davies**, and **Merav Parter**. Simple, deterministic, constant-round coloring in the congested clique. *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 309–318. ACM, 2020 DOI (6).

[28] **Abhik Kumar Das and Sriram Vishwanath**. On finite alphabet compressive sensing. *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 5890–5894. IEEE, 2013 DOI (16).

[29] **Devdatt P. Dubhashi and Alessandro Panconesi**. Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press, 2009 (15).

[30] **Paul Erdős**, **Arthur L Rubin**, and **Herbert Taylor**. Choosability in graphs. *Congr. Numer*, 26(4):125–157, 1979 (3).

[31] **Joan Feigenbaum**, **Sampath Kannan**, **Andrew McGregor**, **Siddharth Suri**, and **Jian Zhang**. Graph distances in the data-stream model. *SIAM J. Comput.* 38(5):1709–1727, 2008 DOI (3, 8).

[32] **Joan Feigenbaum**, **Sampath Kannan**, **Andrew McGregor**, **Siddharth Suri**, and **Jian Zhang**. On graph problems in a semi-streaming model. *Theor. Comput. Sci.* 348(2-3):207–216, 2005 DOI (2, 4, 8).

[33] **Manuela Fischer**, **Magnús M. Halldórsson**, and **Yannic Maus**. Fast distributed Brooks' theorem. *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2567–2588. SIAM, 2023 DOI (6).

[34] **Mohsen Ghaffari**, **Juho Hirvonen**, **Fabian Kuhn**, and **Yannic Maus**. Improved distributed Delta-coloring. *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 427–436. ACM, 2018 (3, 6).

[35] **Péter Hajnal and Endre Szemerédi**. Brooks coloring in parallel. *SIAM J. Discret. Math.* 3(1):74–80, 1990 DOI (3, 6).

[36] **Philip Hall**. On representatives of subsets. *Classic Papers in Combinatorics*:58–62, 1987 (15).

[37] **Bjarni V. Halldórsson**, **Magnús M. Halldórsson**, **Elena Losievskaja**, **and Mario Szegedy**. Streaming algorithms for independent sets in sparse hypergraphs. *Algorithmica*, 76(2):490–501, 2016 DOI (5).

[38] **Magnús M. Halldórsson**, **Fabian Kuhn**, **Yannic Maus**, **and Tigran Tonoyan**. Efficient randomized distributed coloring in CONGEST. *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1180–1193. ACM, 2021 DOI (15).

[39] **Magnús M. Halldórsson**, **Fabian Kuhn**, **Alexandre Nolin**, **and Tigran Tonoyan**. Near-optimal distributed degree+1 coloring. *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 450–463. ACM, 2022 DOI (2, 6, 8).

[40] **Magnús M. Halldórsson**, **Xiaoming Sun**, **Mario Szegedy**, **and Chengu Wang**. Streaming and communication complexity of clique approximation. *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2012 DOI (5).

[41] **David G. Harris**, **Johannes Schneider**, **and Hsin-Hao Su**. Distributed $(\Delta+1)$-coloring in sublogarithmic rounds. *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 465–478. ACM, 2016 DOI (15).

[42] **T. S. Jayram**, **Ravi Kumar**, **and D. Sivakumar**. Two applications of information complexity. *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 673–682. ACM, 2003 DOI (8, 56).

[43] **Mauricio Karchmer and Joseph Naor**. A fast parallel algorithm to color a graph with delta colors. *J. Algorithms*, 9(1):83–91, 1988 DOI (3, 6).

[44] **Howard J. Karloff**. An NC algorithm for Brooks' theorem. *Theor. Comput. Sci.* 68(1):89–103, 1989 DOI (3, 6).

[45] **Howard J. Karloff**, **Siddharth Suri**, **and Sergei Vassilvitskii**. A model of computation for mapreduce. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010 DOI (5).

[46] **Alexandr V. Kostochka**, **Landon Rabern**, **and Michael Stiebitz**. Graphs with chromatic number close to maximum degree. *Discret. Math.* 312(6):1273–1281, 2012 DOI (6).

[47] **László Lovász**. Three short proofs in graph theory. *Journal of Combinatorial Theory, Series B*, 19(3):269–271, 1975 (3, 6).

[48] **Andrew McGregor**. Graph stream algorithms: a survey. *SIGMOD Rec.* 43(1):9–20, 2014 DOI (5, 8).

[49] **LS Melnikov and VG Vizing**. New proof of Brooks' theorem. *Journal of Combinatorial Theory*, 7(4):289–290, 1969 (3).

[50] **Michael Molloy and Bruce Reed**. Graph colouring and the probabilistic method, volume 23. Springer Science & Business Media, 2002 (3, 6, 15, 16, 58).

[51] **Michael Molloy and Bruce A. Reed**. A bound on the strong chromatic index of a graph. *J. Comb. Theory, Ser. B*, 69(2):103–109, 1997 DOI (7, 35, 37).

[52] **Michael Molloy and Bruce A. Reed**. Colouring graphs when the number of colours is almost the maximum degree. *J. Comb. Theory, Ser. B*, 109:134–195, 2014 DOI (6).

[53] **Alessandro Panconesi and Aravind Srinivasan**. Improved distributed algorithms for coloring and network decomposition problems. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 581–592. ACM, 1992 DOI (3, 6).

[54] **Alessandro Panconesi and Aravind Srinivasan**. The local nature of Δ-coloring and its algorithmic applications. *Comb.* 15(2):255–280, 1995 DOI (3, 6).

[55] **Merav Parter**. $(Delta+1)$ coloring in the congested clique model. *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, 160:1–160:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018 DOI (15).

[56] **Landon Rabern**. A different short proof of Brooks' theorem. *Discussiones Mathematicae Graph Theory*, 34(3):633–634, 2014 (3).

[57] **Landon Rabern**. Yet another proof of Brooks' theorem. *Discrete Mathematics*:113261, 2022 (3).

[58] **Jaikumar Radhakrishnan and Saswata Shannigrahi**. Streaming algorithms for 2-coloring uniform hypergraphs. *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, volume 6844 of *Lecture Notes in Computer Science*, pages 667–678. Springer, 2011 DOI (2).

[59] **Bruce A. Reed**. ω, Δ, and χ. *J. Graph Theory*, 27(4):177–212, 1998 DOI (6, 15).

[60] **Bruce A. Reed**. A strengthening of Brooks' theorem. *J. Comb. Theory, Ser. B*, 76(2):136–149, 1999 DOI (6).

[61] **Alexander A. Sherstov**. Approximating the AND-OR tree. *Theory Comput.* 9:653–663, 2013 DOI (55).

[62] **San Skulrattanakulchai**. Delta-list vertex coloring in linear time. *Algorithm Theory - SWAT 2002, 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002 Proceedings*, volume 2368 of *Lecture Notes in Computer Science*, pages 240–248. Springer, 2002 DOI (3, 6).

[63] **Michael Stiebitz and Bjarne Toft**. Brooks's theorem, *Topics in Chromatic Graph Theory*, pages 36–55. Cambridge University Press, 2015 (3).

[64] **Michel Talagrand**. Concentration of measure and isoperimetric inequalities in product spaces. *Publications Mathématiques de l'Institut des Hautes Etudes Scientifiques*, 81:73–205, 1995 (58).

Appendix

A. Proofs of the Impossibility Results

We present the formal proofs of our impossibility results, alluded to Section 2.1, in this appendix. The first is in the query model, and shows that it is essentially impossible to do better than the trivial algorithm that learns the entire graph. The second is in the streaming setting, and shows that it is essentially impossible to do better than to store the entire graph in a slightly non-standard model where edges of the input can appear more than once in the stream.

Notation: We will use a boldface $\mathbf{x}$ to denote a vector (or a bit string), and x_i to index it.

A.1 Sublinear-Time Algorithms

In this section, we show that there is no algorithm in the general query model that solves Δ-coloring in $o(n\Delta)$ queries, via a reduction from the AND-OR-ONE$_{t,m}$ problem.

We assume that the vertices of the graph $G = (V, E)$ are *known*, as is the maximum degree Δ. The general query model supports the following queries on G:

- Degree queries: Given a vertex $v \in V$, output $\deg(v)$.
- Neighbor queries: Given a vertex v, and an index $i \in [\Delta]$, output the i-th neighbor of v, or $\perp$ if $i > \deg(v)$.
- Pair queries: Given two vertices u and $v \in V$, output whether $\{u, v\}$ is an edge in G or not.

The OR problem on N bits is: Given query access to a bit-string $\mathbf{x} \in \{0,1\}^N$, determine whether there exists an i such that $x_i = 1$. By query access, we mean that the algorithm can ask for any $i \in \{N\}$ whether x_i is 0 or 1. It is well known that the randomized query complexity $R(\text{OR}_N)$ is $\Omega(N)$ (see [20]). We will reduce the following promise version of the problem to Δ-coloring (which is also known to have randomized query complexity $\Omega(N)$):

PROBLEM A.1 (OR-ONE$_N$). *Given query access to a string $\mathbf{x} \in \{0,1\}^N$ such that the Hamming weight of $\mathbf{x}$ is at most 1, determine whether there is an index i such that $x_i = 1$.*

Let $N = \binom{n}{2}$. The reduction from OR-ONE$_N$ to Δ-coloring is the following: For a bit-string $\mathbf{x} \in \{0,1\}^N$, we will define a graph G on the vertex set $U = \{u_1, \ldots, u_n\} \cup V = \{v_1, \ldots, v_n\}$. We index $\mathbf{x}$ as $x_{i,j}$ for $1 \leqslant i < j \leqslant n$, and add edges to G as follows:

- If $x_{i,j} = 0$, add the internal edges $\{u_i, u_j\}$ and $\{v_i, v_j\}$.

— If $x_{i,j} = 1$, add the crossing edges $\{u_i, v_j\}$ and $\{v_i, u_j\}$.

Note that the graph G is $(n-1)$-regular (so $\Delta = n-1$). Now, if $\mathbf{x} = 0^N$, then the sets U and V have no edges between them, and G is just two copies of K_n, and hence not Δ-colorable. If, on the other hand, $x_{i,j} = 1$ for one pair (i, j) then G is two copies of K_n minus an edge, connected by a pair of cross edges, and by Brooks' Theorem G is Δ-colorable. To finish the reduction, we need to show that we can simulate each of the queries of the general query model on G with at most a single query on $\mathbf{x}$:

— Degree queries: We always return $n-1$, without looking at any bit of $\mathbf{x}$.

— Neighbor queries: To get the j-th neighbor of u_i (or v_i), we need to look at the bit:

$$\begin{cases} x_{j,i} & \text{if } j < i \\ x_{i,j+1} & \text{if } i \leqslant j \leqslant n-1 \end{cases}$$

— Pair queries: Assume without loss of generality that the query is for the pair (u_i, v_j) such that $i < j$. Then we can answer it by just looking at the bit $x_{i,j}$.

And hence, a $o(n^2)$ query algorithm for Δ-coloring a graph on $2n$ vertices implies a $o(N)$ query algorithm for the OR-ONE problem on $N = \binom{n}{2}$ bits. Note that we can pad an arbitrary instance of OR-ONE$_N$ to an instance we can reduce with at most a constant blowup in size, since $(n+1)^2 \leqslant 2n^2$ for all n large enough.

LEMMA A.2. *The randomized query complexity of Δ-coloring a graph on n vertices is $\Omega(n^2)$ for some choice of $\Delta = \Theta(n)$.*

We can take this idea further: Suppose instead that we have $\Theta(n/\Delta)$ instances of OR-ONE on $\binom{\Delta+1}{2}$ bits. In particular, define the AND-OR-ONE problem:

PROBLEM A.3 (AND-OR-ONE$_{t,m}$). *Given query access to a set of t strings $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^m$, compute:*

$$\bigwedge_{i=1}^{t} \text{OR-ONE}_m(\mathbf{x}_i).$$

Then it is well known that the randomized query complexity of AND-OR-ONE$_{t,m}$ is $\Omega(tm)$ (see [21], [61]). We will take an instance of AND-OR-ONE with $m = \binom{\Delta+1}{2}$ and $t \in \mathbb{N}$—setting $n = t \cdot 2m$—and reduce it to Δ-coloring a graph on $2n$ vertices. In particular, the graph will just be the (disjoint) union of the graphs formed by reducing each of the OR-ONE$_m$ instances in the fashion described above. Note that the graph will be Δ-regular. It is immediate that a $o(n\Delta)$-query algorithm for Δ-coloring implies a $o(t \cdot \Delta^2) = o(tm)$-query algorithm for AND-OR-ONE, and we have a contradiction.

LEMMA A.4. *The randomized query complexity of Δ-coloring a graph on n vertices is $\Omega(n\Delta)$ for all choices of $100 \leqslant \Delta < n/100$.*

A.2 Streaming Algorithms on Repeated-Edge Streams

In this section, we will show that any $O(1)$ pass algorithm for Δ-coloring on graph streams with repeated edges needs $\Omega(n\Delta)$ space to color a graph on n vertices and maximum degree Δ. In particular, we will reduce the TRIBES problem from communication complexity to the $\text{CLIQUE}_{\Delta+1}$ problem. We start by defining the $\text{TRIBES}_{m,n}$ problem.

PROBLEM A.5 ($\text{TRIBES}_{m,n}$). *In the $\text{TRIBES}_{m,n}$ problem, Alice and Bob receive m vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \{0,1\}^n$ and $\mathbf{y}_1, \ldots, \mathbf{y}_m \in \{0,1\}^n$ respectively. They want to compute the function:*

$$\bigwedge_{k=1}^{m} \text{DISJ}_n(\mathbf{x}_k, \mathbf{y}_k).$$

Where DISJ_n is the standard disjoint function that is true iff its inputs differ in all of their bits.

In words, the problem is just to solve m instances of DISJ_n, and return true only if all of them are disjoint. By a result of [42], the randomized communication complexity of $\text{TRIBES}_{m,n}$ is $\Omega(mn)$. We will show a low-communication protocol for $\text{TRIBES}_{m,n}$ assuming a small-space algorithm for $\text{CLIQUE}_{\Delta+1}$, which is defined as follows:

PROBLEM A.6 ($\text{CLIQUE}_{\Delta+1}$). *Given a graph G as a stream, determine whether or not G contains a $(\Delta+1)$-clique.*

First, we will do a warm-up reduction, from DISJ_N to CLIQUE, which simply asks if the input graph is a clique. Suppose that $N = \binom{n}{2}$ for some n,[16] and we have an instance $\mathbf{x}, \mathbf{y} \in \{0,1\}^N$ of DISJ_N. Then we can relabel the indices to (u, v) such that $1 \leqslant u < v \leqslant n$. Let A be the set $\{(i,j) \mid x_{i,j} = 1\}$, and B be the same for $\mathbf{y}$. Define a graph G on the vertex set $V = \{v_1, \ldots, v_n\}$ as follows: Add the edge $\{v_i, v_j\}$ to G iff $\{i,j\}$ is in $\overline{A} \cup \overline{B}$. Note that A and B are disjoint if and only if $\overline{A} \cup \overline{B}$ is the entire universe, which is equivalent to G being a clique.

Assume that we have a constant-pass $o(n^2)$ space algorithm for Δ-coloring (and hence for CLIQUE). Then here is a low-communication protocol for DISJ_N: Alice will form "half" the stream by taking the edge set $\overline{A}$, and run the streaming algorithm for CLIQUE on it, and communicate the state of the algorithm (using $o(n^2)$ bits) to Bob. Bob will then continue running the algorithm on his "half" of the stream (the edge set $\overline{B}$), and hence finish one pass of the algorithm over the edges of G. If there are additional passes required, Bob will communicate the state of the algorithm to Alice (again, using $o(n^2)$ bits) and the process will repeat. After a constant number of passes, the streaming algorithm will decide whether G is a clique, and hence if A and B are disjoint, having used $o(n^2) \cdot O(1)$ communication. And hence we have shown the following lemma:

LEMMA A.7. *There is no constant pass $o(n^2)$ space streaming algorithm for* CLIQUE.

16 This is easily achieved by blowing up the universe by at most a constant factor.

We can use the same idea when starting with an instance of $\textsc{tribes}_{m,k}$ where $m = \binom{\Delta+1}{2}$, and $k \in \mathbb{N}$[17] to rule out a $o(n\Delta)$ space algorithm for $\textsc{clique}_{\Delta+1}$ (and hence Δ-coloring). In particular, define the k-partite graph G on the vertex sets $V_1, \ldots, V_k$, such that each $G(V_i, E_i)$ is the graph that is a clique iff $\mathbf{x}_i$ and $\mathbf{y}_i$ are disjoint.

Then supposing we have a constant pass $o(n\Delta)$ space algorithm for $\textsc{clique}_{\Delta+1}$, we can use exactly the same low communication protocol as before: Alice and Bob each construct half of the graph stream, and to complete a pass exchange the entire state of the algorithm twice. This implies a $o(n\Delta) = o(k\Delta^2) = o(mk)$ communication protocol for $\textsc{tribes}_{m,k}$, and we have the contradiction we desire.

LEMMA A.8. *There is no constant pass $o(n\Delta)$ space streaming algorithm for Δ-coloring.*

The crucial observation is that in the stream created in the communication protocol, the edges of G can *repeat*. In particular, if the bit (i, j) is zero in both $\mathbf{x}$ and $\mathbf{y}$, both Alice and Bob add the edge $\{v_i, v_j\}$ to their halves of the stream. Suppose that our algorithm for Δ-coloring (and hence $\textsc{clique}$) is only required to work on graph streams with no repeated edges. Then the low communication protocol breaks down completely, and hence it is actually possible to solve the Δ-coloring problem in $o(n\Delta)$ space.

A technical remark: Note that both of the lower-bounds we prove are for the problem which tests whether a graph is Δ-colorable, but they also apply to the promise version of the problem where the graph is guaranteed to be Δ-colorable, and the task is to output a coloring. In particular, suppose we have a Δ-coloring algorithm $\mathcal{A}$ on a graph stream, then we can use it for the reduction from $\textsc{disj}_N$ as follows:

- Run $\mathcal{A}$ on the input stream for the graph G with edge set $\overline{A} \cup \overline{B}$, communicating between Alice and Bob as before.
- If $\mathcal{A}$ fails to output a coloring, then the sets A and B are disjoint (w.h.p.).
- If $\mathcal{A}$ outputs a coloring, then we need to *test* it. In particular, if the coloring is improper, there exists an edge in G that is monochromatic. Alice and Bob can independently test their halves of the edges for such an edge, and if they don't find one, the coloring is valid, and hence A and B are disjoint.

A similar argument shows that the query lower-bound applies to the promise version too.

17 And setting $n = k \cdot (\Delta + 1)$.

B. Missing Proofs from Section 5

In this appendix, we show all the results whose proofs we skipped in Section 5. The common factor between these results is that they are all small modifications of previous work and are presented here only for completeness.

Preliminaries: We use a standard form of Talagrand's inequality [64] as specified in [50]. A function $f(x_1, \ldots, x_n)$ is called c-**Lipschitz** iff changing any x_i can affect the value of f by at most c. Additionally, f is called r-**certifiable** iff whenever $f(x_1, \ldots, x_n) \geqslant s$, there exist at most $r \cdot s$ variables $x_{i_1}, \ldots, x_{i_{r \cdot s}}$ so that knowing the values of these variables certifies $f \geqslant s$.

PROPOSITION B.1 (Talagrand's inequality; cf. [50]). *Let $X_1, \ldots, X_m$ be m independent random variables and $f(X_1, \ldots, X_m)$ be a c-Lipschitz function; then for any $t \geqslant 1$,*

$$\Pr\left(|f - \mathbb{E}[f]| > t\right) \leqslant 2\exp\left(-\frac{t^2}{2c^2 \cdot m}\right).$$

Moreover, if f is additionally r-certifiable, then for any $b \geqslant 1$,

$$\Pr\left(|f - \mathbb{E}[f]| > b + 30c\sqrt{r \cdot \mathbb{E}[f]}\right) \leqslant 4\exp\left(-\frac{b^2}{8c^2 r\,\mathbb{E}[f]}\right).$$

B.1 From Section 5.1

LEMMA B.2 (Re-statement of Lemma 5.2). *For every sparse vertex $v \in V_{\text{sparse}}$, in the partial coloring C_1 of* `one-shot-coloring`*,*

$$\mathsf{avail}_{C_1}(v) > (\deg(v) - \mathsf{coldeg}_{C_1}(v)) + \frac{\varepsilon^2 \cdot \Delta}{2\alpha}$$

with high probability, where the randomness is only over the choice of the lists $L_1(v)$.

PROOF. We consider the random variable gap that counts the colors assigned (by x) to *at least* two neighbors of v, and retained (in C_1) by *all* of them. Note that if gap is large, we are in good shape: since each color it counts increases the number of colored neighbors of v by at least 2, while decreasing the number of colors available to v by 1. And hence our aim is to show that gap is large with high probability.

We do this in a roundabout manner. First, we lower bound the expectation of gap with that of the random variable gap', which counts the number of colors assigned to *exactly* two vertices in $N(v)$, and retained by both of them. The main reason for this is that $\mathbb{E}[\mathsf{gap}']$ is easy to calculate, and large enough for our purposes. In particular, let F be the set of non-edges between the neighbors of v, that is

$$F = \{\{u, w\} \subset N(v) \mid \{u, w\} \notin E(G)\}.$$

Then since v is sparse, $|F| \geqslant \varepsilon^2 \cdot \Delta^2/2$ (by Definition 3.3).

For a color $c \in [\Delta]$ and a non-edge $f = \{u, w\} \in F$, let $\mathsf{gap}'_{c,f}$ indicate the event:

— $x(u) = x(w) = c$,
— no other vertex in $N(v) \cup \{v\}$ receives the color c from x, and
— no vertex in $N(u) \cup N(w)$ receives c.

By definition, $\mathsf{gap}' = \sum_{c,f} \mathsf{gap}'_{c,f}$. We have:

$$\Pr\left(\mathsf{gap}'_{c,f}\right) \geqslant \frac{1}{\Delta^2} \cdot \left(1 - \frac{1}{\Delta}\right)^{3\Delta} \geqslant \frac{1}{\Delta^2} \cdot \exp\left(-\frac{2}{\Delta} \cdot 3\Delta\right) = \frac{1}{e^6 \cdot \Delta^2}.$$

Where the first inequality follows because $|N(u) \cup N(w) \cup N(v)| \leqslant 3\Delta$, and the second because

$$\exp\left(-\frac{2}{\Delta}\right) \leqslant 1 - \frac{2}{\Delta} + \frac{2}{\Delta^2} \leqslant 1 - \frac{1}{\Delta}.$$

By linearity of expectation,

$$\mathbb{E}[\mathsf{gap}] \geqslant \mathbb{E}[\mathsf{gap}'] \geqslant \underbrace{\Delta}_{\text{choose } c} \cdot \underbrace{\frac{\varepsilon^2 \cdot \Delta^2}{2}}_{\text{choose } f} \cdot \frac{1}{e^6 \cdot \Delta^2} = \frac{\varepsilon^2 \cdot \Delta}{2e^6}.$$

Next, we want to show that gap is concentrated around its expectation, and for once, Chernoff does not suffice. Define the random variable assign which counts the number of colors *assigned* (by x) to at least two vertices in $N(v)$, and the random variable lose which counts the number of colors assigned to at least two vertices in $N(v)$, and lost by *any* of them. Then clearly $\mathsf{gap} = \mathsf{assign} - \mathsf{lose}$. We will show that assign and lose are both concentrated around their means, and this implies that gap is too.

First, note that assign depends only on the assignment $x(w)$ for all w in the neighborhood of v, that is:

$$\mathsf{assign}\colon \prod_{w \in N(v)} [\Delta] \to \mathbb{N}.$$

Further, it is 1-Lipschitz – changing $x(w)$ from c to c' can:

— Make it so c occurs only once (instead of twice) in $N(v)$, decreasing assign by 1.
— Make it so c' occurs twice (instead of once) in $N(v)$, increasing assign by 1.

And hence the net change to assign from changing $x(w)$ is at most 1 in absolute value. Then by the first part of Talagrand's Inequality (Proposition B.1, with $m = \Delta$, $c = 1$, and $t = \frac{\varepsilon^2 \cdot \Delta}{20e^6}$) we have:

$$\Pr\left(|\mathsf{assign} - \mathbb{E}[\mathsf{assign}]| \geqslant \frac{\varepsilon^2 \cdot \Delta}{20e^6}\right) \leqslant 2 \cdot \exp\left(-\frac{2\varepsilon^4 \cdot \Delta^2}{800e^{12} \cdot \Delta}\right).$$

Which for $\Delta = \Omega(\log^5 n)$ (the Ω hides a monstrous constant), is $1/\mathrm{poly}(n)$. Note that our choice of t is 1/10-th of the lower bound on the expected value of gap.

The same argument does not work for lose—the random variable depends on the 2-hop neighborhood of v, which has size roughly Δ^2, and hence the bound we get above is too weak. However, note that in addition to being 2-Lipschitz, lose is also 3-certifiable. More concretely, let W denote the 2-hop neighborhood of v, then:

— Changing $x(w)$ for some w in W can change the contribution of at most 2 colors to lose: the old and the new color assigned by x to w. Hence lose is 2-Lipschitz.

— For any s, to get lose $\geqslant s$, we need to set $x(\cdot)$ for three vertices (two neighbors of v, and one of their common neighbors) to i, for $i \in [s]$.

Then by second part of Talagrand's Inequality (Proposition B.1, with $c = 2$, $r = 3$, $b = \frac{\varepsilon^2 \cdot \Delta}{20e^6}$):

$$\Pr\left(|\mathsf{lose} - \mathbb{E}[\mathsf{lose}]| > \frac{\varepsilon^2 \cdot \Delta}{20e^6} + 60\sqrt{3 \cdot \mathbb{E}[\mathsf{lose}]}\right) \leqslant 4 \cdot \exp\left(-\left(\frac{\varepsilon^2 \cdot \Delta}{20e^6}\right)^2 \cdot \frac{1}{48 \cdot \mathbb{E}[\mathsf{lose}]}\right).$$

Crudely, we upper bound lose (and hence $\mathbb{E}[\mathsf{lose}]$) by $\Delta/2$, to give us:

$$\Pr\left(|\mathsf{lose} - \mathbb{E}[\mathsf{lose}]| > \frac{\varepsilon^2 \cdot \Delta}{20e^6} + 60\sqrt{3/2 \cdot \Delta}\right) \leqslant 4 \cdot \exp\left(-\frac{\varepsilon^4}{9600e^{12}}\Delta\right).$$

Which with $\Delta = \Omega(\log^5 n)$ (and Ω doing an even braver job) is $1/\mathrm{poly}(n)$.

Taking the union of the bad events (i.e. either assign or lose deviates too much from its mean), and applying the triangle inequality, we have that with high probability:

$$\mathsf{gap} > \mathbb{E}[\mathsf{gap}] - \frac{\varepsilon^2 \cdot \Delta}{10e^6} - O(\sqrt{\Delta}).$$

And hence combining with the lower bound on $\mathbb{E}[\mathsf{gap}]$ we found earlier, $\mathsf{gap} > \frac{\varepsilon^2 \cdot \Delta}{2000}$ with high probability. ■

B.2 From Section 5.3

LEMMA B.3 (Re-statement of Lemma 5.8). *With high probability, there exists a proper partial Δ-coloring C that is an extension of C_2 and colors all remaining vertices in V_{sparse} using only the colors in the lists $L_3(v)$ for sparse vertices $v \in V_{\text{sparse}}$ (and thus the randomness is also only over these lists).*

PROOF. Let C initially be the coloring C_2. We will color the vertices of V_{sparse} greedily by updating C. That is, we iterate over V_{sparse} in arbitrary order, and if a vertex v is uncolored in C, we pick a color in $L_3(v)$ which does not conflict with any of its neighbors in H, and set $C(v)$ to it. Using the randomness of $L_3(v)$, and conditioning on the high probability event of Lemma 5.2:

CLAIM B.4. *With high probability, for each vertex $v \in V_{\text{sparse}}$, and any partial coloring C that extends C_1, there exists a color $c \in L_3(v)$ that is not used in $N_H(v)$ by C.*

Proof. We would like to show that $\mathsf{Avail}_C(v)$ and $L_3(v)$ have a nonzero intersection with high probability. Since v is sparse, by Lemma 5.2 we have:

$$\mathsf{avail}_C(v) > (\deg(v) - \mathsf{coldeg}_{C_1}(v)) + \frac{\varepsilon^2 \cdot \Delta}{2\alpha}$$

Which implies that even if the entire neighborhood of v is colored by C, there are still $\frac{\varepsilon^2}{2\alpha} \cdot \Delta$ colors available for v to use.

Since each color from $\mathsf{Avail}_C(v)$ is sampled into $L_3(v)$ independently with probability $\frac{100\alpha \cdot \log n}{\varepsilon^2 \cdot \Delta}$,

$$\Pr\left(\mathsf{Avail}_C(v) \cap L_3(v) = \emptyset\right) \leqslant \left(1 - \frac{100\alpha \cdot \log n}{\varepsilon^2 \cdot \Delta}\right)^{\frac{\varepsilon^2 \cdot \Delta}{2\alpha}} \leqslant \exp\left(-\frac{100\alpha \cdot \log n}{\varepsilon^2 \cdot \Delta} \cdot \frac{\varepsilon^2 \cdot \Delta}{2\alpha}\right) < n^{-50},$$

concluding the proof. ◆

And hence the greedy algorithm can color V_{sparse}. Note that this algorithm can be implemented efficiently using the information we gathered in Section 4. In particular, since C only assigns colors from $L(v)$ to v, it is enough to check for conflicts in H while coloring from $L_3(v)$. ■

B.3 From Section 5.4

LEMMA B.5 (Re-statement of Lemma 5.10). *For a holey almost-clique K, and any partial Δ-coloring C outside K, there exists, with high probability, a coloring C' which extends C to K such that $C'(v) \in L_4(v)$ for all $v \in K$.*

The main thing we want to exploit is that K has a lot of non-edges, since we can assign the same color to both endpoints of a non-edge. One easy way to do this for many non-edges simultaneously is to generate a "matching" of non-edges, which we do via the following algorithm:

It follows immediately that M is a matching; we will show that for holey cliques, M is *large* with constant probability.

LEMMA B.6. *If K is a holey clique, Algorithm 8 produces a matching of size $\ell = \frac{t}{10^6 \varepsilon \Delta}$ with probability at least $1/2$.*

PROOF. We define:

- $\mathsf{Present}(c)$ as the set of non-edges present in F when we encounter color c in line (ii) of Algorithm 8. Let $\mathsf{present}(c)$ denote $|\mathsf{Present}(c)|$.
- A color c is *successful* if we add assign it to a non-edge during Algorithm 8.

Note that the number of successful colors is exactly the size of M, and hence we would like to show that many colors succeed. To do so, we show that $\mathsf{present}(\cdot)$ is large for many colors. In particular, we say that a color c is *heavy* if it has $\mathsf{present}(c) \geqslant t/2$, and have the following claim:

Input: The input graph G, an almost-clique K, the partial coloring C, and the list $L_{4,i}(v)$ for each v.

(i) Initialize $C' \leftarrow C$.

(ii) Let $F = \{f_1, \ldots, f_t\}$ be the set of non-edges in K.

(iii) For each color $c \in [\Delta]$: If there is a non-edge $f = \{u, v\} \in F$ such that $c \in L_{4,i}(u) \cap L_{4,i}(v)$, and c is not used in the neighborhood of u or v, then:

- Assign $C'(u) \leftarrow c$ and $C'(v) \leftarrow c$.
- Add f to M.
- Remove all non-edges incident on f from F.

Algorithm 8. The `colorful-matching` algorithm.

CLAIM B.7. *There are at least $\Delta/2$ heavy colors.*

PROOF. For a non-edge $f = \{u, v\}$, define $\mathsf{Blocked}(f)$ to be the set of colors used by C to color the neighbors of u or v outside K. By property item iii). of Definition 3.4, the number of neighbors of u or v (and hence the number of colors used by C to color them) is at most $20\varepsilon\Delta$. As a result, at the beginning of the algorithm:

$$\sum_{f \in F} |\mathsf{Blocked}(f)| \leqslant t \cdot 20\varepsilon\Delta.$$

This means that on average, each color occurs in $\mathsf{Blocked}(f)$ for at most

$$t \cdot 20\varepsilon\Delta \cdot 1/\Delta = 20\varepsilon t$$

non-edges f. By Markov's Inequality, there are at most $\Delta/2$ colors c which occur in $\mathsf{Blocked}(f)$ for more than $40\varepsilon t$ non-edges f. Hence there are at least $\Delta/2$ colors c which are *not blocked* for at least $t - 40\varepsilon t \geqslant 9/10 \cdot t$ non-edges in F—at the beginning of the algorithm.

How many of these non-edges remain in $\mathsf{Present}(c)$ when we look at c? Upon adding the non-edge $\{u, v\}$ to M, we remove all non-edges incident on u or v from F. Since each $u \in K$ has at most $10\varepsilon\Delta$ non-neighbors in K (property ii). of Definition 3.4), each non-edge added to M removes at most $20\varepsilon\Delta$ edges from $\mathsf{Present}(c)$. Because the algorithm has already succeeded if $|M|$ becomes larger than ℓ, the number of edges removed from $\mathsf{Present}(c)$ is at most $\ell \cdot 20\varepsilon\Delta$ in total, which is $< t/10$. Hence there are at least $\Delta/2$ colors c with $\mathsf{present}(c) \geqslant 8/10 \cdot t \geqslant t/2$. ■

Regrouping, we have shown that there are many colors in $[\Delta]$ that can be assigned to many non-edges in F. Next, we would like to show that for each of these heavy colors c, the probability that c is in sampled by both endpoints of a non-edge in $\mathsf{Present}(c)$ is high.

CLAIM B.8. *For a heavy color c, $\Pr(c \text{ is successful}) \geqslant 10^{-5} \cdot t/\varepsilon\Delta^2$.*

Proof. The color c is successful if there is least one non-edge $\{u, v\} \in \mathsf{Present}(c)$ such that $c \in L_{4,i}(u) \cap L_{4,i}(v)$. Using the inclusion exclusion principle, and cutting out terms of "order" more than 2:

$$\Pr(c \text{ is successful}) \geqslant \sum_{\{u,v\} \in \mathsf{Present}(c)} \Pr\left(c \in L_{4,i}(u) \cap L_{4,i}(v)\right) - \sum_{f,g \in \mathsf{Present}(c)} \Pr\left(c \in \bigcap_{w \in f \cup g} L_{4,i}(w)\right).$$

The first term is easy to compute exactly: c belongs to both lists with probability q^2, so

$$\sum_{\{u,v\} \in \mathsf{Present}(c)} \Pr\left(c \in L_{4,i}(u) \cap L_{4,i}(v)\right) = \mathsf{present}(c) \cdot q^2.$$

For the second term, we have to consider two cases:

— $|f \cup g| = 3$: The probability that c belongs to $L_{4,i}(w)$ for 3 vertices w is q^3. After picking $f = \{u, v\}$ from $\mathsf{Present}(c)$, there are at most $20\varepsilon\Delta$ choices for the third vertex, and hence the total contribution of terms of this type is at most

$$\mathsf{present}(c) \cdot 20\varepsilon\Delta \cdot q^3.$$

— $|f \cup g| = 4$: The probability that c belongs to the list of 4 vertices is q^4. And there are at most $\mathsf{present}(c)^2$ ways to pick a 2-set of non-edges. So the total contribution of these terms is at most

$$\mathsf{present}(c)^2 \cdot q^4.$$

Adding everything up, we get:

$$\begin{aligned}
\Pr(c \text{ is successful}) &\geqslant \mathsf{present}(c) \cdot q^2 - \mathsf{present}(c) \cdot 10\varepsilon\Delta \cdot q^3 - \mathsf{present}(c)^2 \cdot q^4 \\
&\geqslant 9/10 \cdot \mathsf{present}(c) \cdot q^2 - \mathsf{present}(c)^2 \cdot q^4 \\
&\qquad \text{(since } q = \tfrac{1}{100\sqrt{\varepsilon}\Delta} \text{ and thus } 10\varepsilon\Delta \cdot q < 1/10 \text{ for } \varepsilon < 1\text{)} \\
&\geqslant \mathsf{present}(c) \cdot q^2 \cdot \left(9/10 - 20\varepsilon\Delta^2 \cdot q^2\right) \\
&\qquad \text{(since } \mathsf{present}(c) \leqslant |F| \leqslant 20\varepsilon\Delta^2 \text{ by property } ii\text{). of Definition 3.4)} \\
&\geqslant 8/10 \cdot \mathsf{present}(c) \cdot q^2 \qquad \text{(since } q = \frac{1}{100\sqrt{\varepsilon}\Delta}, 20\varepsilon\Delta^2 \cdot q^2 = 1/200\text{)} \\
&\geqslant \frac{t}{10^5 \cdot \varepsilon\Delta^2},
\end{aligned}$$

concluding the proof. ◆

Finally, we are ready to prove the lemma itself. Let $\theta = 10^{-5} \cdot t/\varepsilon\Delta^2$ (the RHS of Claim B.8); note that $\theta < 1$ because $t \leqslant |F| \leqslant 20\varepsilon\Delta^2$. Let Z be a random variable with the binomial distribution $\mathcal{B}(\Delta/2, \theta)$. Note that we can couple each Bernoulli trial used to determine Z with a heavy color succeeding – since there are at least $\Delta/2$ of them, they succeed with probability at least θ, and two different colors are independent of each other. Hence Z is a lower bound for $|M|$. We follow our usual formula:

$$\mathbb{E}[Z] = \Delta/2 \cdot \theta = 10^{-5} \cdot t/2\varepsilon\Delta.$$

And with an application of Chernoff Bound (Proposition 3.2, with $\delta = 1/2$):

$$\Pr\left(Z < 1/2 \cdot \mathbb{E}[Z]\right) \leqslant 2\exp\left(-\frac{(1/2)^2 \cdot t}{10^5 \cdot 2\varepsilon\Delta \cdot (2+1/2)}\right) \leqslant 2\exp\left(-\frac{10^7 \cdot \varepsilon\Delta}{10^6 \cdot 2\varepsilon\Delta}\right) \leqslant 2e^{-5} < 1/2.$$

And hence with probability at least $1/2$,

$$|M| \geqslant Z \geqslant 10^{-5} \cdot t/8\varepsilon\Delta > 10^{-6} \cdot t/\varepsilon\Delta = \ell.$$

This concludes the proof. ∎

Now, since we run Algorithm 8 for β independent sets lists $\{L_{4,i}(v) \mid v \in V\}$, we get a non-edge matching M of size $\ell = t/\varepsilon\Delta$ from one of the runs with high probability. We keep the coloring assigned to this largest non-edge edge matching, and hence have the following lemma:

LEMMA B.9. *Suppose K is a holey almost-clique, with $t \geqslant 10^7 \cdot \varepsilon\Delta$ non-edges inside it. Then for any coloring C outside K, with high probability there is an extension C' of C which:*

- *Colors $2 \cdot \frac{t}{10^6 \cdot \varepsilon\Delta}$ vertices of K.*
- *Uses only $\frac{t}{10^6 \cdot \varepsilon\Delta}$ colors inside K, and further for each $v \in K$ that it colors, uses a color from $L_4(v)$.*

Let us pause for a moment, and consider why we did all this work. Lemma B.9 tells us that Algorithm 8 colors *some* number of vertices in K, using only half that many unique colors. As in the case of sparse vertices, this creates enough of a gap between available colors and the remaining degree of each vertex in K such that an available color is sampled in $L_4(v)$ with high probability. This is exactly what we need to prove Lemma 5.10.

PROOF OF LEMMA 5.10. Let C' be the partial coloring obtained from Lemma B.9. Then we define the base and sampled palette graphs $\mathcal{G}_{\texttt{Base}}$ and $\mathcal{G}_{\texttt{Sample}}$ (Definitions 3.8 and 3.9) with:

- $\mathcal{L}$ as the set of vertices of K *not* colored by C'.
- $\mathcal{R}$ as the set of colors *not* used by C' in K.
- $S(v) = L_4^*(v)$ for each $v \in \mathcal{L}$.

We will show that $\mathcal{G}_{\texttt{Base}}$ satsifies the conditions in Lemma 3.10, and hence $\mathcal{G}_{\texttt{Sample}}$ has an $\mathcal{L}$-perfect matching with high probability, and hence C' can be extended to $(L_4 \cup L_4^*)$-color all of K. In the same order as the lemma:

(i) For $m := \mathcal{L}$, $m \leqslant |\mathcal{R}| \leqslant 2m$: To get the first inequality, we rewrite $|\mathcal{L}|$ and $|\mathcal{R}|$ in terms of K and the vertices and colors removed by C':

$$|\mathcal{L}| = |K| - \frac{2t}{10^6 \cdot \varepsilon\Delta}, |\mathcal{R}| = \Delta - \frac{t}{10^6 \cdot \varepsilon\Delta}.$$

Note that if $|K| \geqslant \Delta + 1 + k$, the number of non-edges inside K (that is, t) is at least $k \cdot \Delta$. Then the number of vertices removed far outstrips the number of colors, in particular:

$$|\mathcal{L}| = \Delta + 1 + k - \frac{2k}{10^6 \cdot \varepsilon} \leqslant \Delta - k, |\mathcal{R}| = \Delta - \frac{k}{10^6 \cdot \varepsilon}.$$

Which makes $|\mathcal{L}| \leqslant |\mathcal{R}|$ for $k \geqslant 1$, because $\frac{1}{10^6 \cdot \varepsilon} > 2$. On the other hand, if $K = \Delta + 1$, since $t \geqslant 10^7 \cdot \varepsilon\Delta$,

$$|\mathcal{R}| - |\mathcal{L}| = \frac{10^7 \cdot \varepsilon\Delta}{10^6 \cdot \varepsilon\Delta} - 1 \geqslant 9.$$

To get the second inequality, first note that $m \geqslant 2/3 \cdot \Delta$. This is because the maximum number of non-edges in K is $(2\Delta) \cdot (10\varepsilon\Delta) = 20\varepsilon\Delta^2$ (by property ii). of Definition 3.4), and hence the number of vertices in $\mathcal{L}$ is at least

$$(1 - 5\varepsilon)\Delta - \frac{40\varepsilon\Delta^2}{10^6 \cdot \varepsilon\Delta} \geqslant 2/3 \cdot \Delta.$$

Then since $|R| \leqslant \Delta$ we are done.

(ii) Each vertex $v \in \mathcal{L}$ has $\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant 2/3 \cdot m$: Each vertex v in $\mathcal{L}$ may have up to $10\varepsilon\Delta$ edges out of K (in G), hence blocking $10\varepsilon\Delta$ colors in $\mathcal{R}$ for v. Since all the remaining colors are available to v,

$$\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |R| - 10\varepsilon\Delta \geqslant m - 15\varepsilon \cdot m \geqslant 2/3 \cdot m.$$

Where the second inequality follows from part (i).

(iii) For every set $A \subset \mathcal{L}$ of size $|A| \geqslant m/2$, we have $\sum_{v \in A} \deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |A| \cdot m - m/4$; recall that for any $v \in \mathcal{L}$:

$$\deg_{\mathcal{G}_{\texttt{Base}}}(v) \geqslant |R| - (\Delta + 1) + |K| - \overline{\deg}_K(v).$$

Let $T := \frac{t}{10^6 \cdot \varepsilon\Delta} \geqslant 10$. Summing the inequality above over $v \in A$, we get:

$$\begin{aligned}
\sum_{v\in A} \deg_{\mathcal{G}_{\texttt{Base}}}(v) &\geqslant \sum_{v\in A} \Big(|R| - (\Delta+1) + |K| - \overline{\deg}_K(v)\Big) \\
&= \sum_{v\in A} (\underbrace{\Delta - T}_{|R|} - (\Delta+1) + \underbrace{|\mathcal{L}| + 2T}_{|K|} - \overline{\deg}_K(v)) \\
&= \sum_{v\in A} \Big(|L| - \overline{\deg}_K(v) + T - 1\Big) \\
&= |A| \cdot (m + T - 1) - \underbrace{\sum_{v\in A} \overline{\deg}_K(v)}_{\leqslant 2t} > |A| \cdot m.
\end{aligned}$$

Where the last inequality follows from

$$(T-1) \cdot |A| \geqslant T/2 \cdot |A| \geqslant \frac{tm}{10^6 \cdot 4\varepsilon\Delta} \geqslant \frac{t\Delta}{10^6 \cdot 6\varepsilon\Delta} \geqslant 5t.$$

And now by the promised application of Lemma 3.10, $\mathcal{G}_{\texttt{Sample}}$ has an $\mathcal{L}$-perfect matching, and we are done. ∎

2023:9 TheoretiCS