Graph Coloring via Degeneracy in Streaming and Other Space-Conscious Models

Suman K. Bera
University of California at Santa Cruz, CA, USA

Amit Chakrabarti
Dartmouth College, Hanover, NH, USA
http://www.cs.dartmouth.edu/~ac

Prantar Ghosh
Dartmouth College, Hanover, NH, USA

Abstract

We study the problem of coloring a given graph using a small number of colors in several well-established models of computation for big data. These include the data streaming model, the general graph query model, the massively parallel communication (MPC) model, and the CONGESTED-CLIQUE and the LOCAL models of distributed computation. On the one hand, we give algorithms with sublinear complexity, for the appropriate notion of complexity in each of these models. Our algorithms color a graph G using $\kappa(G) \cdot (1 + o(1))$ colors, where $\kappa(G)$ is the degeneracy of G: this parameter is closely related to the arboricity $\alpha(G)$. As a function of $\kappa(G)$ alone, our results are close to best possible, since the optimal number of colors is $\kappa(G) + 1$. For several classes of graphs, including real-world "big graphs," our results improve upon the number of colors used by the various $(\Delta(G) + 1)$-coloring algorithms known for these models, where $\Delta(G)$ is the maximum degree in G, since $\Delta(G) \geqslant \kappa(G)$ and can in fact be arbitrarily larger than $\kappa(G)$.

On the other hand, we establish certain lower bounds indicating that sublinear algorithms probably cannot go much further. In particular, we prove that any randomized coloring algorithm that uses at most $\kappa(G) + O(1)$ colors would require $\Omega(n^2)$ storage in the one pass streaming model, and $\Omega(n^2)$ many queries in the general graph query model, where n is the number of vertices in the graph. These lower bounds hold even when the value of $\kappa(G)$ is known in advance; at the same time, our upper bounds do not require $\kappa(G)$ to be given in advance.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Dynamic graph algorithms; Theory of computation → Sketching and sampling; Theory of computation → Lower bounds and information complexity

Keywords and phrases Data streaming, Graph coloring, Sublinear algorithms, Massively parallel communication, Distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.11

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/pdf/1905.00566.pdf.

Funding This work was supported in part by NSF under award CCF-190773 and by the Simons Institute for the Theory of Computing, which hosted the second author while the work was in progress.

Acknowledgements The authors gratefully acknowledge several helpful discussions with Sepehr Assadi (especially those that called to their attention a nuance with the Congested Clique algorithm) and Deeparnab Chakrabarty.

1 Introduction

Graph coloring is a fundamental topic in combinatorics and the corresponding algorithmic problem of coloring an input graph with a small number of colors is a basic and heavily studied problem in computer science. It has numerous applications, e.g., in scheduling [67, 49, 48], air traffic flow management [15], frequency assignment in wireless networks [8, 56], and register allocation [21, 26, 22]. More recently, vertex coloring has been used to compute seed vertices in social networks that are then expanded to detect community structures in the network [54].

Given an n-vertex graph $G = (V, E)$, the task is to assign colors to the vertices in V so that no two adjacent vertices get the same color. Doing so with the minimum possible number of colors – called the chromatic number, $\chi(G)$ – is famously hard: it is NP-hard to even approximate $\chi(G)$ to a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [31, 68, 45]. In the face of this hardness, it is algorithmically interesting to color G with a possibly suboptimal number of colors depending upon tractable parameters of G. One such simple parameter is Δ, the maximum degree: a trivial greedy algorithm colors G with $\Delta + 1$ colors in linear time.

We study graph coloring in a number of space-constrained and data-access-constrained settings, including the data streaming model, a query model, and certain distributed computing models. As expected, coloring using the optimal number of colors is hard in these models. Abboud et al. [1] show that coloring an n-vertex graph G with $\chi(G)$ colors in the p-pass streaming setting requires $\Omega(n^2/p)$ space, and checking c-colorability for $3 \leqslant c < n$ requires $\Omega((n-c)^2/p)$ space. In such constrained settings, even finding a coloring with "about Δ" colors is a fairly nontrivial problem that has been studied from various angles in a flurry of research over the last decade [5, 10, 23, 24, 40, 59]. In a recent breakthrough (awarded Best Paper at SODA 2019), Assadi, Chen, and Khanna [5] gave sublinear algorithms for $(\Delta + 1)$-coloring an input graph in such models.

In this work, we focus on colorings that use "about κ" colors, where $\kappa = \kappa(G)$ is the *degeneracy* of G, a parameter that improves upon Δ. It is defined as follows: $\kappa = \min\{k :$ every induced subgraph of G has a vertex of degree at most $k\}$. Every graph is $(\kappa + 1)$-colorable. Clearly, $\kappa \leqslant \Delta$ and can be much smaller than Δ for sparse graphs and real-world graphs; see Table 2. Thus, our aim is to use fewer colors when $(\Delta + 1)$-coloring might be wasteful. A closely related parameter is $\alpha(G)$, the arboricity of the graph, which is the minimum number of forests into which the set of edges of G can be partitioned. It is known that $\alpha \leqslant \kappa < 2\alpha$. A number of works has studied the $O(\alpha)$-coloring problem in distributed computing models [11, 12, 35, 36, 47]. However, to the best of our knowledge, such coloring algorithms were unknown in the data streaming and graph query models.

There is a simple greedy algorithm that runs in linear time and produces a $(\kappa+1)$-coloring; see Section 3. However, just as before, when processing a massive graph under the constraints of either the space-bounded streaming model or certain distributed computing models, the inherently sequential nature of the greedy algorithm makes it infeasible. We overcome this barrier with a very simple framework: decompose the graph into smaller subgraphs so as to store all the blocks in our limited memory, and then run the greedy algorithm on each block. We show that this basic framework (with careful implementation in the respective models) suffices for obtaining the colorings we seek.

On the other hand, we give a number of lower bounds showing that, despite its simplicity, our algorithmic framework does about as good a job as sublinear algorithms can. In particular, no randomized algorithm can achieve $(\kappa + O(1))$-colorings without spending $\Omega(n^2)$ space in the streaming model or $\Omega(n^2)$ queries in the query model.

1.1 Our Results and Techniques

Algorithms. We give new graph coloring algorithms, parametrized by the degeneracy κ, in the following models:

(1) the data streaming model, where the input is a stream of edge insertions and deletions (i.e., a dynamic graph stream) resulting in the eventual graph to be colored and we are limited to a work space of $\widetilde{O}(n)$ bits[1], the so-called semi-streaming setting [32];

(2) the general graph query model [38], where we may access the graph using only neighbor queries (what is the ith neighbor of x?) and pair queries (are x and y adjacent?);

(3) the massively parallel communication (MPC) model, where each of a large number of memory-limited processors holds a sublinear-sized portion of the input data and computation proceeds using rounds of communication;

(4) the congested clique model of distributed computation, where there is one processor per vertex holding that vertex's neighborhood information and each round allows each processor to communicate $O(\log n)$ bits to a specific other processor; and

(5) the LOCAL model of distributed computation, where there is one processor per vertex holding that vertex's neighborhood information and each round allows each processor to send an arbitrary amount of information to all its neighbors.

Table 1 Summary of our algorithmic results and basic comparison with most related previous work. In the result marked (*), we require that $\kappa = \omega(\log^2 n)$.

Model	Colors	Complexity Parameters	Reference
Streaming	$\Delta + 1$	$\widetilde{O}(n)$ space, $\widetilde{O}(n\sqrt{\Delta})$ post-processing time	[5]
(one pass)	$\kappa + o(\kappa)$	$\widetilde{O}(n)$ space, $\widetilde{O}(n)$ post-processing time	this paper
Query	$\Delta + 1$	$\widetilde{O}(n^{3/2})$ queries	[5]
	$\kappa + o(\kappa)$	$\widetilde{O}(n^{3/2})$ queries	this paper
MPC	$\Delta + 1$	$O(1)$ rounds, $O(n \log^3 n)$ bits per processor	[5]
	$\kappa + o(\kappa)$	$O(1)$ rounds, $O(n \log^2 n)$ bits per processor	this paper
Congested Clique	$\Delta + 1$	$O(1)$ rounds	[23]
	$O(\kappa)$	$O(1)$ rounds	[36]
	$\kappa + o(\kappa)^\star$	$O(1)$ rounds	this paper
LOCAL	$O(\kappa n^{1/k})$	$O(k)$ rounds, for $k \in \left[\omega(\log\log n), O(\sqrt{\log n})\right]$	[47]
	$O(\kappa n^{1/k} \log n)$	$O(k)$ rounds, for $k \in \left[\omega(\sqrt{\log n}), o(\log n)\right]$	this paper

Table 1 summarizes our algorithmic results and provides, in each case, a basic comparison with the most related result from prior work. We give an elaborate account of the related works in Section 2.

As we have noted, $\kappa \leqslant \Delta$ in every case; indeed, κ could be arbitrarily better than Δ as shown by the example of a star graph, where $\kappa = 1$ whereas $\Delta = n - 1$. From a practical standpoint, it is notable that in many real-world large graphs drawn from various application domains – such as social networks, web graphs, and biological networks – the parameter κ is often *significantly* smaller than Δ. See Table 2 for some concrete numbers. That said, $\kappa + o(\kappa)$ is, in general, mathematically incomparable with $\Delta + 1$.

[1] The $\widetilde{O}(\cdot)$ notation hides factors polylogarithmic in n.

Table 2 Statistics of several large real-world graphs taken from the application domains of social networks, web graphs, and biological networks, showing that the degeneracy, κ, is often significantly smaller than the maximum degree, Δ. Source: `http://networkrepository.com` [61].

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Δ	κ
soc-friendster	66M	2B	5K	305
fb-uci-uni	59M	92M	5K	17
soc-livejournal	4M	28M	3K	214
soc-orkut	3M	106M	27K	231
web-baidu-baike	2M	18M	98K	83
web-hudong	2M	15M	62K	529

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Δ	κ
web-wikipedia2009	2M	5M	3K	67
web-google	916K	5M	6K	65
bio-mouse-gene	43K	14M	8K	1K
bio-human-gene1	22K	12M	8K	2K
bio-human-gene2	14K	9M	7K	2K
bio-WormNet-v3	16K	763K	1K	165

A key contribution here is a conceptual idea and a corresponding technical lemma underlying all our algorithms. We show that every graph admits a "small" sized *low degeneracy partition* (LDP), which is a partition of its vertex set into "few" blocks such that the subgraph induced by each block has low degeneracy, roughly logarithmic in n. Moreover, such an LDP can be computed by a very simple and distributed randomized algorithm: for each vertex, choose a "color" independently and uniformly at random from a suitable-sized palette (this is not to be confused with the eventual graph coloring we seek; this random assignment is most probably not a proper coloring of the graph). The resulting color classes define the blocks of such a partition, with high probability. Theorem 8, the LDP Theorem, makes this precise. Given an LDP, a generic graph coloring algorithm is to run the aforementioned greedy algorithm on each block, using distinct palettes for the distinct blocks. We obtain algorithms achieving our claimed results by suitably implementing this generic algorithm in each computational model.

Lower Bounds. Recall that a graph with degeneracy κ admits a proper $(\kappa+1)$-coloring. As Table 1 makes clear, there are several space-conscious $(\Delta+1)$-coloring algorithms known; perhaps we could aim for improved algorithms that provide $(\kappa+1)$-colorings? We prove that this is not possible in sublinear complexity in either the streaming or the query model. In fact, we prove more. We show that distinguishing n-vertex graphs of degeneracy κ from those with chromatic number $\kappa+2$ requires $\Omega(n^2)$ space in the streaming model and $\Omega(n^2)$ queries in the general graph query model. This shows that it is hard to produce a $(\kappa+1)$-coloring and in fact even to determine the value of κ. These results generalize to the problems of producing a $(\kappa+\lambda)$-coloring or estimating the degeneracy up to $\pm\lambda$; the corresponding lower bounds are $\Omega(n^2/\lambda^2)$. Furthermore, the streaming lower bounds hold even in the insertion-only model, where the input stream is simply a listing of the graph's edges in some order; compare this with our upper bound, which works even for dynamic graph streams.

A possible criticism of the above lower bounds for coloring is that they seem to depend on it being hard to *estimate* the degeneracy κ. Perhaps the coloring problem could become easier if κ was given to the algorithm in advance? We prove two more lower bounds showing that this is not so: the same $\Omega(n^2/\lambda^2)$ bounds hold even with κ known *a priori*.

In the full version of this paper [18], we present a "combinatorial" lower bound that addresses a potential criticism of our main algorithmic technique: the LDP. Perhaps a more sophisticated graph-theoretic result, such as the Palette Sparsification Theorem of Assadi et al. (see Section 2), could improve the quality of the colorings obtained? We prove that this is not so: there is no analogous theorem for colorings with "about κ" colors.

2 Related Work and Comparisons

Streaming and Query Models. Assadi et al. [5] give a one-pass streaming $(\Delta+1)$-coloring algorithm that uses $\widetilde{O}(n)$ space (i.e., is semi-streaming) and works on dynamic graph streams. They also give a $(\Delta+1)$-coloring algorithm in the general graph query model that makes $\widetilde{O}(n^{3/2})$ queries. For these, they establish a beautiful *Palette Sparsification Theorem*: a combinatorial result saying that choosing a random $O(\log n)$-sized palette from $\{1,\ldots,\Delta+1\}$ for each vertex allows a compatible list coloring. While we do not get a similarly tight combinatorial result – there isn't one, as we just noted – we do achieve faster post-processing time ($\widetilde{O}(n)$ versus their $\widetilde{O}(n\sqrt{\Delta})$) in the streaming setting and have a simpler post-processing algorithm (greedy offline versus matching-based) in the query setting, while keeping other complexity parameters the same (Table 1). These wins come at the price of a less tight result – $(1+o(1))\kappa$ colors instead of the combinatorially optimal $\kappa+1$ – but of course our streaming and query lower bounds show that such slack is necessary. Also, as noted before, we often have $\kappa \ll \Delta$ (Table 2).

For streaming lower bounds, Abboud et al. [1] show that coloring a graph G with $\chi(G)$ colors requires $\Omega(n^2/p)$ space in p passes. They also show that deciding c-colorability for $3 \leqslant c < n$ (that might be a function of n) takes $\Omega((n-c)^2/p)$ space in p passes. Furthermore, any streaming algorithm that distinguishes between $\chi(G) \leqslant 3c$ and $\chi(G) \geqslant 4c$ must use $\Omega(n^2/pc^2)$ space. Another recent work on coloring in the streaming model is Radhakrishnan et al. [60], which studies the problem of 2-coloring an n-uniform hypergraph.

In the query model, there are a number of works studying basic graph problems [39, 57, 25] but, to the best of our knowledge, Assadi et al. were the first to study graph coloring in this sense. Also, to the best of our knowledge, there was no previously known algorithm for $O(\alpha)$-coloring in the semi-streaming and graph query settings, whereas here we obtain $(\kappa+o(\kappa))$-colorings; recall the bound $\kappa \leqslant 2\alpha-1$.

MPC and Congested Clique Models. The MapReduce framework [27] is extensively used in distributed computing to process massive data sets. Beame, Koutris, and Suciu [16] defined the Massively Parallel Communication (MPC) model to abstract out key theoretical features of MapReduce; it has since become a widely used setting for designing and analyzing big data algorithms, especially for graph problems. Another well studied model for distributed graph algorithms is Congested Clique [50]. Behnezhad et al. [17] show that Congested Clique is equivalent to the "semi-MPC model," defined as MPC with $O(n\log n)$ bits of memory per machine, thanks to simulations in both directions preserving the round complexity.

Harvey et al. [41] gave a $(\Delta+o(\Delta))$-coloring algorithm in the MapReduce model; it can be simulated in MPC using $O(1)$ rounds and $O(n^{1+c})$ space per machine for some constant $c>0$. The aforementioned paper of Assadi et al. [5] gives an $O(1)$-round MPC algorithm for $(\Delta+1)$-coloring using $O(n\log^3 n)$ bits of space per machine. Because this space usage is $\omega(n\log n)$, the equivalence result of Behnezhad et al. [17] does not apply and this doesn't lead to an $O(1)$-round Congested Clique algorithm. In contrast, our MPC algorithm can be made to use only $O(n\log n)$ bits per machine and $\kappa+o(\kappa)$ colors for graphs with $\kappa=\omega(\log^2 n)$, and therefore leads to such a Congested Clique algorithm. Chang et al. [23] gave an $O(\sqrt{\log\log n})$-round MPC algorithm with $o(n)$ space per machine and $\widetilde{O}(m)$ space in total. Using the improved network decomposition results by Rozhon and Ghaffari [62], this round complexity can be reduced to $O(\log\log\log n)$. We, however, focus on the quasi-linear memory per machine regime.

Graph coloring has recently garnered considerable attention in the Congested Clique model. Parter [58] gave a $(\Delta+1)$-coloring algorithm using $O(\log\log\Delta\cdot\log^\star\Delta)$ rounds, later improved to $O(\log^\star\Delta)$ by Parter and Su [59]. Chang et al. [23] have recently improved this to $O(1)$ rounds. They use similar but more involved graph partitioning techniques than us, as is probably necessary for a stringent $(\Delta+1)$-coloring. For low-degeneracy graphs, our algorithm uses fewer colors than all these algorithms while achieving the best possible asymptotic round complexity ($O(1)$). Parallel to our work, Ghaffari and Sayyadi [36] gave an $O(1)$-round algorithm for the $O(\alpha)$-coloring problem. Their analysis suggests that they obtain a $(c\alpha)$-coloring algorithm, where the constant $c > 10$. On the other hand, we get a tight $(1+o(1))\kappa$-coloring. Recall, again, that $\kappa \leqslant 2\alpha-1$ (Fact 2). Hence, we have an arguably simpler algorithmic framework achieving better results. The main novelty in our techniques lies in choosing degeneracy as the key parameter (instead of arboricity, which could lead to results looser by a factor of 2) and in the careful analysis that gives very sharp – not just asymptotic – bounds on the number of colors. Our algorithm (only the Congested Clique implementation), however, needs $\kappa=\omega(\log^2 n)$ or $\kappa=O(1)$ to keep the round complexity constant.

The LOCAL Model. There is a deep body of work on graph coloring in this model. Indeed, graph coloring is one of *the* most central "symmetry breaking" problems in distributed computing. We refer the reader to the monograph by Barenboim and Elkin [13] for an excellent overview of the state of the art. Here, we shall briefly discuss only a few results closely related to our contribution.

There is a long line of work on fast $(\Delta+1)$-coloring in the LOCAL model, in the deterministic as well as the randomized setting [55, 10, 33, 51, 44, 3, 63, 14] culminating in sublogarithmic time solutions due to Harris [40] and Chang et al. [24]. Barenboim and Elkin [11, 12] studied fast distributed coloring algorithms that may use far fewer than Δ colors: in particular, they gave algorithms that use $O(\alpha)$ colors and run in $O(\alpha^\varepsilon\log n)$ time on graphs with arboricity at most α. Recall again that $\kappa\leqslant 2\alpha-1$, so that a 2α-coloring always *exists*. They also gave a faster $O(\log n)$-time algorithm using $O(\alpha^2)$ colors. Further, they gave a family of algorithms that produce an $O(t\alpha^2)$-coloring in $O(\log_t n+\log^\star n)$, for every t such that $2\leqslant t\leqslant O(\sqrt{n/\alpha})$. Our algorithm for the LOCAL model builds on this latter result.

Kothapalli and Pemmaraju [47] focused on arboricity-dependant coloring using very few rounds. They gave a randomized $O(k)$-round algorithm that uses $O(\alpha n^{1/k})$ colors for $2\log\log n\leqslant k\leqslant\sqrt{\log n}$ and $O(\alpha^{1+1/k}n^{1/k+3/k^2}2^{-2^k})$ colors for $k<2\log\log n$. We extend their result to the range $k\in\left[\omega(\sqrt{\log n}),\, o(\log n)\right]$, using $O(\alpha n^{1/k}\log n)$ colors.

Ghaffari and Lymouri [35] gave a randomized $O(\alpha)$-coloring algorithm that runs in time $O(\log n\cdot\min\{\log\log n,\log\alpha\})$ as well as an $O(\log n)$-time algorithm using $\min\{(2+\varepsilon)\alpha+O(\log n\log\log n),\, O(\alpha\log\alpha)\}$ colors, for any constant $\varepsilon>0$. However, their technique does not yield a sublogarithmic time algorithm, even at the cost of a larger palette.

The LDP Technique. As mentioned earlier, our algorithmic results rely on the concept of a low degeneracy partition (LDP) that we introduce in this work. Some relatives of this idea have been considered before. Specifically, Barenboim and Elkin [13] define a d-defective (resp. b-arbdefective) c-coloring to be a vertex coloring using palette $[c]$ such that every color class induces a subgraph with maximum degree at most d (resp. arboricity at most b). Obtaining such improper colorings is a useful first step towards obtaining proper colorings. They give deterministic algorithms to obtain good arbdefective colorings [12]. However, their algorithms are elaborate and are based on construction of low outdegree acyclic partial orientations of the graph's edges: an expensive step in our space-conscious models.

Elsewhere (Theorem 10.5 of Barenboim and Elkin [13]), they note that a useful defective (not arbdefective) coloring is easily obtained by randomly picking a color for each vertex; this is then useful for computing an $O(\Delta)$-coloring.

Our LDP technique can be seen as a simple randomized method to produce an arbdefective coloring. Crucially, we parametrize our result using degeneracy instead of arboricity and give sharp – not just asymptotic – bounds on the degeneracy of each color class.

The Degeneracy Parameter. The parameter has been studied under several other names, such as *width* [34], *linkage* [46] and *Szekeres-Wilf number* [66]. For a graph G, the number $\kappa(G) + 1$ is often called the *coloring number* of G [28, 65]. It has also been extensively studied as *k-core number* in different areas such as data streaming and parallel computing [29], distributed systems [4], data mining [53], protein networks [6], and social networks[20]. Farach-Colton and Tsai [30] studied the parameter in the streaming model, and gave a one-pass semi-streaming algorithm that approximates the degeneracy of an input graph within a multiplicative factor of $1 + \varepsilon$. Our lower bounds complement this result as we show that computing the degeneracy κ exactly or more generally within a multiplicative factor of $(1 + \kappa^{-(1/2+\gamma)})$, for some constant γ, is not possible in the one-pass semi-streaming setting.

Other Related Work. Other work considers coloring in the setting of *dynamic graph algorithms*: edges are inserted and deleted over time and the goal is to *maintain* a valid vertex coloring of the graph that must be updated quickly after each modification. Unlike in the streaming setting, there is no space restriction. Bhattacharya et al. [19] gave a randomized algorithm that maintains a $(\Delta + 1)$-coloring with $O(\log \Delta)$ expected amortized update time, later improved to $O(1)$ by Henzinger and Peng [43]. Solomon and Wein [64] studied the problem for low-arboricity graphs and gave an $O(\alpha \log^2 n)$-coloring algorithm with $O(\text{poly}(\log \log n))$ update time. Recently, Henzinger et al. [42] designed an $O(\alpha \log n)$-coloring algorithm with $O(\log^2 n)$ update time. Barba et al. [9] gave tradeoffs between the number of colors used and update time. However, the techniques in these works do not seem to apply in the streaming setting due to fundamental differences in the models.

Estimating the arboricity of a graph in the streaming model is a well studied problem. McGregor et al. [52] gave a one pass $(1+\varepsilon)$-approximation algorithm to estimate the arboricity of graph using $\widetilde{O}(n)$ space. Bahmani et al. [7] gave a matching lower bound. Our lower bounds for estimating degeneracy are quantitatively much larger but they call for much tighter estimates.

3 Preliminaries

Throughout this paper, graphs are simple, undirected, and unweighted. In considering a graph coloring problem, the input graph will usually be called G and we will put $n = |V(G)|$. The notation "$\log x$" stands for $\log_2 x$. For an integer k, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$.

For a graph G, we define $\Delta(G) = \max\{\deg(v) : v \in V(G)\}$. We say that G is k-degenerate if every induced subgraph of G has a vertex of degree at most k. For instance, every forest is 1-degenerate and an elementary theorem says that every planar graph is 5-degenerate. The *degeneracy* $\kappa(G)$ is the smallest k such that G is k-degenerate. The *arboricity* $\alpha(G)$ is the smallest r such that the edge set $E(G)$ can be partitioned into r forests. When the graph G is clear from the context, we simply write Δ, κ, and α, instead of $\Delta(G)$, etc.

We note two useful facts: the first is immediate and the second is an easy exercise.

▶ **Fact 1.** *If an n-vertex graph has degeneracy κ, then it has at most κn edges.*

▶ **Fact 2.** *In every graph, the degeneracy κ and arboricity α satisfy $\alpha \leqslant \kappa \leqslant 2\alpha - 1$.*

In analyzing our algorithms, it will be useful to consider certain *vertex orderings* of graphs and their connection with the notion of degeneracy, given by Lemma 5 below. Although the lemma is folklore, it is crucial to our analysis, so we include a proof for completeness.

▶ **Definition 3.** *An* ordering *of G is a list consisting of all its vertices (equivalently, a total order on $V(G)$). Given an ordering $\lhd$, for each $v \in V(G)$, the* ordered neighborhood $N_{G,\lhd}(v) := \{w \in V(G) : \{v, w\} \in E(G), v \lhd w\}$*, i.e., the set of neighbors of v that appear* after *v in the ordering. The* ordered degree $\operatorname{odeg}_{G,\lhd}(v) := |N_{G,\lhd}(v)|$.

▶ **Definition 4.** *A* degeneracy ordering *of G is an ordering produced by the following algorithm: starting with an empty list, pick a minimum degree vertex v (breaking ties arbitrarily), append v to the end of the list, and recurse on $G - v$ if it is nonempty.*

▶ **Lemma 5.** *G is k-degenerate iff there exists an ordering $\lhd$ such that $\operatorname{odeg}_{G,\lhd}(v) \leqslant k$ for all $v \in V(G)$.*

Proof. Suppose that G is k-degenerate. Let $\lhd = (v_1, \ldots, v_n)$ be a degeneracy ordering. Then, for each i, $\operatorname{odeg}_{G,\lhd}(v_i)$ is the degree of v_i in the induced subgraph $H := G \setminus \{v_1, \ldots, v_{i-1}\}$. By definition, H has a vertex of degree at most k, so v_i, being a minimum degree vertex in H, must have degree at most k.

On the other hand, suppose that G has an ordering $\lhd$ such that $\operatorname{odeg}_{G,\lhd}(v) \leqslant k$ for all $v \in V(G)$. Let H be an induced subgraph of G. Let v be the leftmost (i.e., smallest) vertex in $V(H)$ according to $\lhd$. Then all neighbors of v in H in fact lie in $N_{G,\lhd}(v)$, so $\deg_H(v) \leqslant \operatorname{odeg}_{G,\lhd}(v) \leqslant k$. Therefore, G is k-degenerate. ◀

A c-coloring of a graph G is a mapping $\psi \colon V(G) \to [c]$; it is said to be a *proper coloring* if it makes no edge monochromatic: $\psi(u) \neq \psi(v)$ for all $\{u, v\} \in E(G)$. The smallest c such that G has a proper c-coloring is called the *chromatic number* $\chi(G)$. By considering the vertices of G one at a time and coloring greedily, we immediately obtain a proper $(\Delta + 1)$-coloring. This idea easily extends to degeneracy-based coloring.

▶ **Lemma 6.** *Given unrestricted ("offline") access to an input graph G, we can produce a proper $(\kappa + 1)$-coloring of G in linear time.*

Proof. Construct a degeneracy ordering $(v_1, \ldots, v_n)$ of G and then greedily color the vertices one by one in the order $(v_n, \ldots, v_1)$. Given a palette of size $\kappa + 1$, by the "only if" direction of Lemma 5, there will always be a free color for a vertex. ◀

Of course, the simple algorithm above is not implementable directly in "sublinear" settings, such as space-bounded streaming algorithms, query models, or distributed computing models. Nevertheless, we shall use it on suitably constructed subgraphs of our input graph.

We shall use the following form of the Chernoff bound.

▶ **Fact 7.** *Let X be a sum of mutually independent indicator random variables. Let μ and δ be real numbers such that $\mathbb{E}X \leqslant \mu$ and $0 \leqslant \delta \leqslant 1$. Then, $\Pr[X \geqslant (1 + \delta)\mu] \leqslant \exp\left(-\mu\delta^2/3\right)$.*

4 A Generic Framework for Coloring Algorithms

In this section, we give a generic framework for graph coloring that we later instantiate in various computational models. As a reminder, our focus is on graphs G with a nontrivial upper bound on the degeneracy $\kappa = \kappa(G)$. Each such graph *admits* a proper $(\kappa + 1)$-coloring; our focus will be on obtaining a proper $(\kappa + o(\kappa))$-coloring efficiently.

As a broad outline, our framework calls for coloring G in two phases. The first phase produces a *low degeneracy partition* (LDP) of G: it partitions $V(G)$ into a "small" number of parts, each of which induces a subgraph that has "low" degeneracy. This step can be thought of as preprocessing and it is essentially free (in terms of complexity) in each of our models. The second phase properly colors each part, using a small number of colors, which is possible because the degeneracy is low. In later sections, we shall see that the low degeneracy allows this second phase to be efficient in each of the models we consider.

4.1 A Low Degeneracy Partition

In this phase of our coloring framework, we assign each vertex a color chosen uniformly at random from $[\ell]$, these choices being mutually independent, where ℓ is a suitable parameter. For each $i \in [\ell]$, let G_i denote the subgraph of G induced by vertices colored i. We shall call each G_i a *block* of the vertex partition given by $(G_1, \ldots, G_\ell)$. The next theorem, our main technical tool, provides certain guarantees on this partition given a suitable choice of ℓ.

▶ **Theorem 8** (LDP Theorem). *Let G be an n-vertex graph with degeneracy κ. Let $k \in [1, n]$ be a "guess" for the value of κ and let $s \geqslant Cn \log n$ be a sparsity parameter, where C is a sufficiently large universal constant. Put*

$$\ell = \left\lceil \frac{2nk}{s} \right\rceil , \quad \lambda = 3\sqrt{\kappa \ell \log n}\,, \tag{1}$$

and let $\psi \colon V(G) \to [\ell]$ be a uniformly random coloring of G. For $i \in [\ell]$, let G_i be the subgraph induced by $\psi^{-1}(i)$. Then, the partition $(G_1, \ldots, G_\ell)$ has the following properties.

- **(i)** *If $k \leqslant 2\kappa$, then w.h.p., for each i, the degeneracy $\kappa(G_i) \leqslant (\kappa + \lambda)/\ell$.*
- **(ii)** *W.h.p., for each i, the block size $|V(G_i)| \leqslant 2n/\ell$.*
- **(iii)** *If $\kappa \leqslant k \leqslant 2\kappa$, then w.h.p., the number of monochromatic edges $|E(G_1) \cup \cdots \cup E(G_\ell)| \leqslant s$.*

In each case, "w.h.p." means "with probability at least $1 - 1/\operatorname{poly}(n)$."

Proof. Notice that when $k \leqslant (C/2) \log n$, the condition $s \geqslant Cn \log n$ results in $\ell = 1$, so the vertex partition is the trivial one-block partition, which obviously satisfies all the properties in the theorem. Thus, in our proof, we may assume that $k > (C/2) \log n$.

We start with Item ii, which is the most straightforward. From Equation (1), we have $\ell \leqslant 4nk/s$, so

$$\frac{n}{\ell} \geqslant \frac{s}{4k} \geqslant \frac{Cn \log n}{4k} \geqslant \frac{C \log n}{4}\,.$$

Each block size $|V(G_i)|$ has binomial distribution $\operatorname{Bin}(n, 1/\ell)$, so a Chernoff bound gives

$$\Pr\left[|V(G_i)| > \frac{2n}{\ell}\right] \leqslant \exp\left(-\frac{n}{3\ell}\right) \leqslant \exp\left(-\frac{C \log n}{12}\right) \leqslant \frac{1}{n^2}\,,$$

for sufficiently large C. By a union bound over the at most n blocks, Item ii fails with probability at most $1/n$.

Items i and iii include the condition $k \leqslant 2\kappa$, which we shall assume for the rest of the proof. By Equation (1) and the bounds $s \geqslant Cn \log n$ and $k > (C/2) \log n$,

$$\ell \leqslant \left\lceil \frac{2k}{C \log n} \right\rceil \leqslant \frac{4k}{C \log n} \leqslant \frac{8\kappa}{C \log n}\,,$$

whence, for sufficiently large C,

$$\lambda \leqslant 3\sqrt{\kappa \cdot \frac{8\kappa}{C \log n} \cdot \log n} \leqslant \kappa\,. \tag{2}$$

We now turn to establishing Item i. Let $\lhd$ be a degeneracy ordering for G. For each $i \in [\ell]$, let $\lhd_i$ be the restriction of $\lhd$ to $V(G_i)$. Consider a particular vertex $v \in V(G)$ and let $j = \psi(v)$ be its color. We shall prove that, w.h.p., $\operatorname{odeg}_{G,\lhd_j}(v) \leqslant (\kappa+\lambda)/\ell$.

By the "only if" direction of Lemma 5, we have $\operatorname{odeg}_{G,\lhd}(v) = |N_{G,\lhd}(v)| \leqslant \kappa$. Now note that

$$\operatorname{odeg}_{G_j,\lhd_j}(v) = \sum_{u \in N_{G,\lhd}(v)} \mathbb{1}_{\{\psi(u)=\psi(v)\}}$$

is a sum of mutually independent indicator random variables, each of which has expectation $1/\ell$. Therefore, $\mathbb{E}\operatorname{odeg}_{G_j,\lhd_j}(v) = \operatorname{odeg}_{G,\lhd}(v)/\ell \leqslant \kappa/\ell$. Since $\lambda \leqslant \kappa$ by Equation (2), we may use the form of the Chernoff bound in Fact 7, which gives us

$$\Pr\left[\operatorname{odeg}_{G_j,\lhd_j}(v) > \frac{\kappa+\lambda}{\ell}\right] \leqslant \exp\left(-\frac{\kappa}{\ell}\frac{\lambda^2}{3\kappa^2}\right) = \exp\left(-\frac{9\kappa\ell\log n}{3\kappa\ell}\right) \leqslant \frac{1}{n^3}\,,$$

where the equality follows from Equation (1). In words, with probability at least $1-1/n^3$, the vertex v has ordered degree at most $(\kappa+\lambda)/\ell$ within its own block. By a union bound, with probability at least $1-1/n^2$, all n vertices of G satisfy this property. When this happens, by the "if" direction of Lemma 5, it follows that $\kappa(G_i) \leqslant (\kappa+\lambda)/\ell$ for every i.

Finally, we take up Item iii, which is now straightforward. Assume that the high probability event in Item i occurs. Then, by Fact 1,

$$|E(G_1) \cup \cdots \cup E(G_\ell)| \leqslant \sum_{i=1}^{\ell} \kappa(G_i)\,|V(G_i)| \leqslant \frac{\kappa+\lambda}{\ell}\sum_{i=1}^{\ell}|V(G_i)| = \frac{n(\kappa+\lambda)}{\ell} \leqslant \frac{2n\kappa}{\ell} \leqslant s\,,$$

where the final inequality uses the condition $\kappa \leqslant k$ and Equation (1). ◀

It will be convenient to encapsulate the guarantees of this theorem in a definition.

▶ **Definition 9.** *Suppose graph G has degeneracy κ. A vertex partition $(G_1, \ldots, G_\ell)$ simultaneously satisfying the degeneracy bound in Item i, the block size bound in Item ii, and the (monochromatic) edge sparsity bound in Item iii in Theorem 8 is called an (ℓ, s, λ)-LDP of G.*

It will turn out that an (ℓ, s, λ)-LDP leads to a proper coloring of G using at most $\kappa+\lambda+\ell$ colors. An instructive setting of parameters is $s = \Theta((n\log n)/\varepsilon^2)$, where ε is either a small constant or a slowly vanishing function of n, such as $1/\log n$. Then, a quick calculation shows that when an accurate guess $k \in [\kappa, 2\kappa]$ is made, Theorem 8 guarantees an LDP that has edge sparsity $s = \widetilde{O}(n)$ and that leads to an eventual proper coloring using $(1+O(\varepsilon))\kappa$ colors. When $\varepsilon = o(1)$, this number of colors is $\kappa + o(\kappa)$.

Recall that the second phase of our coloring framework involves coloring each G_i separately, exploiting its low degeneracy. Indeed, given an (ℓ, s, λ)-LDP, each block G_i *admits* a proper $(\kappa(G_i)+1)$-coloring. Suppose we use a distinct palette for each block; then the total number of colors used is

$$\sum_{i=1}^{\ell}(\kappa(G_i)+1) \leqslant \ell\left(\frac{\kappa+\lambda}{\ell}+1\right) = \kappa+\lambda+\ell\,, \tag{3}$$

as claimed above. Of course, even if our first phase random coloring ψ yields a suitable LDP, we still have to collect each block G_i or at least enough information about each block so as to produce a proper $(\kappa(G_i)+1)$-coloring. How we do this depends on the precise model of computation; see Sections 5 and 6 here and the full version [18] for further instantiations.

4.2 Applications in Various Models

We now turn to the application of the framework in designing graph coloring algorithms in specific models of computation for big data, where the focus is on utilizing space sublinear in the size of the massive input graph. Such models are sometimes termed *space-conscious*.

In Section 5, we discuss the simulation of our framework in the streaming model. We present a semi-streaming algorithm in the dynamic model, meaning that edges can be both inserted and deleted. Our main result is captured in Theorem 10.

▶ **Theorem 10.** *Set $s = \lceil \varepsilon^{-2} n \log n \rceil$, where $\varepsilon > 0$ is a parameter. There is a one-pass algorithm that processes a dynamic (i.e., turnstile) graph stream using $O(\varepsilon^{-2} n \log^4 n)$ bits of space and, with high probability, produces a proper coloring using at most $(1+O(\varepsilon))\kappa$ colors. In particular, taking $\varepsilon = 1/\log n$, it produces a $(\kappa + o(\kappa))$-coloring using $\widetilde{O}(n)$ space. Each edge update is processed in $\widetilde{O}(1)$ time and end-of-stream post-processing takes $\widetilde{O}(n)$ time.*

In Section 6, applying our framework to the general graph query model, we obtain:

▶ **Theorem 11.** *Given query access to a graph G, there is a randomized algorithm that, with high probability, produces a proper coloring of G using $\kappa + o(\kappa)$ colors. The algorithm's worst-case query complexity, running time, and space usage are all $\widetilde{O}(n^{3/2})$.*

Besides this, we obtain algorithmic results in certain distributed models of computation, namely MPC, Congested-Clique, and LOCAL models, where graph coloring is one of the most heavily studied problems. Our results are stated below. See the full version of our paper [18] for the corresponding discussions and proofs.

▶ **Theorem 12.** *There is a randomized $O(1)$-round MPC algorithm that, given an n-vertex graph G, outputs a $(\kappa + o(\kappa))$-coloring of G with high probability. The algorithm uses n processors, each with $O(n \log^2 n)$ bits of memory.*

▶ **Theorem 13.** *There is a randomized $O(1)$-round algorithm in the Congested Clique model that, given a graph G, w.h.p. finds a $(\kappa + O(\kappa^{3/4} \log^{1/2} n))$-coloring. For $\kappa = \omega(\log^2 n)$, this gives a $(\kappa + o(\kappa))$-coloring.*

▶ **Theorem 14.** *There is a randomized distributed algorithm in the LOCAL model that, given an n-vertex graph G, an estimate of its arboricity α up to a constant factor, and a parameter t such that $2 < t \leqslant O(\sqrt{n/\log n})$, produces an $O(t\alpha \log n)$-coloring of G in time $O\left(\log_t n + \log^\star n\right)$.*

5 Streaming Model

We turn to the most intensely studied space-conscious model: the data streaming model. For graph problems, in the basic model, the input is a stream of non-repeated edges that define the input graph G: this is called the *insertion-only* model, since it can be thought of as building up G through a sequence of edge insertions. In the more general *dynamic graph model* or *turnstile model*, the stream is a sequence of edge updates, each update being either

an insertion or a deletion: the net effect is to build up G. Our algorithm below will work in this more general model. Later, in we shall give a corresponding lower bound that will hold even in the insertion-only model (for a lower bound, this is a strength).

We assume that the vertex set $V(G) = [n]$ and the input is a stream σ of at most $m = \text{poly}(n)$ updates to an initially empty graph. An update is a triple (u, v, c), where $u, v \in V(G)$ and $c \in \{-1, 1\}$: when $c = 1$, this token represents an insertion of edge $\{u, v\}$ and when $c = -1$, it represents a deletion. Let $N = \binom{n}{2}$ and $[[m]] = \mathbb{Z} \cap [-m, m]$. It is convenient to imagine a vector $\mathbf{x} \in [[m]]^N$ of edge multiplicities that starts at zero and is updated entrywise with each token. The input graph G described by the stream will be the underlying simple graph, i.e., $E(G)$ will be the set of all edges $\{u, v\}$ such that $x_{u,v} \neq 0$ at the end. We shall say that σ *builds up* $\mathbf{x}$ and G.

Our algorithm makes use of two data streaming primitives, each a *linear sketch*. (We can do away with these sketches in the insertion-only setting; see the end of this section.) The first is a sketch for *sparse recovery* given by a matrix A (say): given a vector $\mathbf{x} \in [[m]]^N$ with sparsity $\|\mathbf{x}\|_0 \leqslant t$, there is an efficient algorithm to reconstruct $\mathbf{x}$ from $A\mathbf{x}$. The second is a sketch for ℓ_0 *estimation* given by a random matrix B (say): given a vector $\mathbf{x} \in [[m]]^N$, there is an efficient algorithm that takes $B\mathbf{x}$ and computes from it an estimate of $\|\mathbf{x}\|_0$ that, with probability at least $1 - \delta$, is a $(1 + \gamma)$-multiplicative approximation. It is known that there exists a suitable $A \in \{0, 1\}^{d \times N}$, where $d = O(t \log(N/t))$, where A has column sparsity $O(\log(N/t))$; see, e.g., Theorem 9 of Gilbert and Indyk [37]. It is also known that there exists a suitable distribution over matrices giving $B \in \{0, 1\}^{d' \times N}$ with $d' = O(\gamma^{-2} \log \delta^{-1} \log N(\log \gamma^{-1} + \log \log m))$. Further, given an update to the ith entry of $\mathbf{x}$, the resulting updates in $A\mathbf{x}$ and $B\mathbf{x}$ can be effected quickly by generating the required portion of the ith columns of A and B.

Algorithm 1 One-Pass Streaming Algorithm for Graph Coloring via Degeneracy.

procedure COLOR(stream σ, integer k) ▷ σ builds up $\mathbf{x}$ and G; $k \in [1, n]$ is a guess for $\kappa(G)$
  choose s, ℓ as in Equation (1) and t, d, d', A, B as in the above discussion
  initialize $\mathbf{y} \in [[m]]^d$ and $\mathbf{z} \in [[m]]^{d'}$ to zero
  foreach $u \in [n]$ **do** $\psi(u) \leftarrow$ uniform random color in $[\ell]$
  foreach token (u, v, c) in σ **do**
    if $\psi(u) = \psi(v)$ **then** $\mathbf{y} \leftarrow \mathbf{y} + cA_{u,v}$; $\mathbf{z} \leftarrow \mathbf{z} + cB_{u,v}$
  if estimate of $\|\mathbf{w}\|_0$ obtained from $\mathbf{z}$ is $> 5s/4$ **then abort**
  $\mathbf{w}' \leftarrow$ result of t-sparse recovery from $\mathbf{y}$ ▷ we expect that $\mathbf{w}' = \mathbf{w}$
  foreach $i \in [\ell]$ **do**
    $G_i \leftarrow$ simple graph induced by $\{\{u, v\} : w'_{u,v} \neq 0 \text{ and } \psi(u) = \psi(v) = i\}$
    color G_i using palette $\{(i, j) : 1 \leqslant j \leqslant \kappa(G_i) + 1\}$; cf. Lemma 6 ▷ net effect is to color G

In our description of Algorithm 1, we use $A_{u,v}$ (resp. $B_{u,v}$) to denote the column of A (resp. B) indexed by $\{u, v\}$. The algorithm's logic results in sketches $\mathbf{y} = A\mathbf{w}$ and $\mathbf{z} = B\mathbf{w}$, where $\mathbf{w}$ corresponds to the subgraph of G consisting of ψ-monochromatic edges only (cf. Theorem 8), i.e., $\mathbf{w}$ is obtained from $\mathbf{x}$ by zeroing out all entries except those indexed by $\{u, v\}$ with $\psi(u) = \psi(v)$. We choose the parameter $t = 2s$, where $s \geqslant Cn \log n$ is the sparsity parameter from Theorem 8, which gives $d = O(s \log n)$; we choose $\gamma = 1/4$ and $\delta = 1/n$, giving $d' = O(\log^3 n)$.

Notice that Algorithm 1 requires a guess for $\kappa := \kappa(G)$, which is not known in advance. Our final one-pass algorithm runs $O(\log n)$ parallel instances of COLOR(σ, k), using geometrically spaced guesses $k = 2, 4, 8 \ldots$. It outputs the coloring produced by the non-aborting run that uses the smallest guess. This leads to this section's main result (restated from Section 4.2).

▶ **Theorem 10.** *Set $s = \lceil \varepsilon^{-2} n \log n \rceil$, where $\varepsilon > 0$ is a parameter. There is a one-pass algorithm that processes a dynamic (i.e., turnstile) graph stream using $O(\varepsilon^{-2} n \log^4 n)$ bits of space and, with high probability, produces a proper coloring using at most $(1 + O(\varepsilon))\kappa$ colors. In particular, taking $\varepsilon = 1/\log n$, it produces a $(\kappa + o(\kappa))$-coloring using $\widetilde{O}(n)$ space. Each edge update is processed in $\widetilde{O}(1)$ time and end-of-stream post-processing takes $\widetilde{O}(n)$ time.*

Proof. The coloring produced is obviously proper. Let us bound the number of colors used. One of the parallel runs of $\textsc{Color}(\sigma, k)$ in Algorithm 1 will use a value $k = k^\star \in (\kappa, 2\kappa]$. We shall prove that, w.h.p., (a) every non-aborting run with $k \leqslant k^\star$ will use at most $(1 + O(\varepsilon))\kappa$ colors, and (b) the run with $k = k^\star$ will not abort.

We start with (a). Consider a particular run using $k \leqslant k^\star$. By Item i of Theorem 8, each G_i has degeneracy at most $(\kappa + \lambda)/\ell$; so if $\mathbf{w}$ is correctly recovered by the sparse recovery sketch (i.e., $\mathbf{w}' = \mathbf{w}$ in Algorithm 1), then each G_i is correctly recovered and the run uses at most $\kappa + \lambda + \ell$ colors, as in Equation (3). Using the values from Equation (1), this number is at most $(1 + O(\varepsilon))\kappa$. Now, if the run does not abort, then the estimate of the sparsity $\|\mathbf{w}\|_0$ is at most $5s/4$. By the guarantees of the ℓ_0-estimation sketch, the true sparsity is at most $(5/4)(5s/4) < 2s = t$, so, w.h.p., $\mathbf{w}$ is indeed t-sparse and, by the guarantees of the sparse recovery sketch, $\mathbf{w}' = \mathbf{w}$. Taking a union bound over all $O(\log n)$ runs, the bound on the number of colors holds for all required runs simultaneously, w.h.p..

We now take up (b). Note that $\|\mathbf{w}\|_0$ is precisely the number of ψ-monochromatic edges in G. By Item iii of Theorem 8, we have $\|\mathbf{w}_0\| \leqslant s$ w.h.p. By the accuracy guarantee of the ℓ_0-estimation sketch, in this run the estimate of $\|\mathbf{w}\|_0$ is at most $5s/4$ w.h.p., so the run does not abort.

The space usage of each parallel run is dominated by the computation of $\mathbf{y}$, so it is $O(d \log m) = O(s \log n \log m) = O(\varepsilon^{-2} n \log^3 n)$, using our setting of s and the assumption $m = \text{poly}(n)$. The claims about the update time and post-processing time follow directly from the properties of a state-of-the-art sparse recovery scheme, e.g., the scheme based on expander matching pursuit given in Theorem 9 of Gilbert and Indyk [37]. ◀

6 Query Model

We now turn to the *general graph query model*, a standard model of space-conscious algorithms for big graphs where the input graph is random-accessible but the emphasis is on the examining only a tiny (ideally, sublinear) portion of it; for general background see Chapter 10 of Goldreich's book [38]. In this model, the algorithm starts out knowing the vertex set $[n]$ of the input graph G and can access G only through the following types of queries.

- A *pair query* $\text{Pair}(\{u, v\})$, where $u, v \in [n]$. The query returns 1 if $\{u, v\} \in E(G)$ and 0 otherwise. For better readability, we shall write this query as $\text{Pair}(u, v)$.
- A *neighbor query* $\text{Neighbor}(u, j)$, where $u \in [n]$ and $j \in [n-1]$. The query returns $v \in [n]$ where v is the jth neighbor of u in some underlying fixed ordering of vertex adjacency lists; if $\deg(v) < j$, so that there does not exist a jth neighbor, the query returns $\perp$.

Naturally, when solving a problem in this model, the goal is to do so while minimizing the number of queries.

6.1 Sublinear Algorithm

By adapting the combinatorial machinery from their semi streaming algorithm, Assadi et al. [5] gave an $\widetilde{O}(n^{3/2})$-query algorithm for finding a $(\Delta+1)$-coloring. Our LDP framework gives a considerably simpler algorithm using $\kappa + o(\kappa)$ colors, where $\kappa := \kappa(G)$. We remark here that $\widetilde{O}(n^{3/2})$ query complexity is optimal (up to polylogarithmic actors), as Assadi et al. [5] proved a matching lower bound for any $(c \cdot \Delta)$-coloring algorithm, for any constant $c > 1$.

▶ **Theorem 11.** *Given query access to a graph G, there is a randomized algorithm that, with high probability, produces a proper coloring of G using $\kappa + o(\kappa)$ colors. The algorithm's worst-case query complexity, running time, and space usage are all $\widetilde{O}(n^{3/2})$.*

Proof. The algorithm proceeds in two stages. In the first stage, it attempts to extract all edges in G through neighbor queries alone, aborting when "too many" queries have been made. More precisely, it loops over all vertices v and, for each v, issues queries $\text{Neighbor}(v, 1), \text{Neighbor}(v, 2), \ldots$ until a query returns $\perp$. If this stage ends up making $3n^{3/2}$ queries (say) without having processed every vertex, then it aborts and the algorithm moves on to the second stage. By Fact 1, if $\kappa \leqslant \sqrt{n}$, then this stage will not abort and the algorithm will have obtained G completely; it can then $(\kappa+1)$-color G (as in Lemma 6) and terminate, skipping the second stage.

In the second stage, we know that $\kappa > \sqrt{n}$. The algorithm now uses a random coloring ψ to construct an (ℓ, s, λ)-LDP of G using the "guess" $k = \sqrt{n}$, with $s = \Theta(\varepsilon^{-2} n \log n)$ and ℓ, λ given by Equation (1). To produce each subgraph G_i in the LDP, the algorithm simply makes all possible queries $\text{Pair}(u, v)$ where $\psi(u) = \psi(v)$. W.h.p., the number of queries made is at most

$$\frac{1}{2}\sum_{i\in[\ell]} |V(G_i)|^2 \leqslant \frac{\ell}{2}\left(\frac{2n}{\ell}\right)^2 \leqslant \frac{2n^2 s}{4nk} = \Theta\left(\frac{n^{3/2}\log n}{\varepsilon^2}\right),$$

where the first inequality uses Item ii of Theorem 8. We can enforce this bound in the worst case by aborting if it is violated.

Clearly, $k \leqslant 2\kappa$, so Item i of Theorem 8 applies and by the discussion after Definition 9, the algorithm uses $(1 + O(\varepsilon))\kappa$ colors. Setting $\varepsilon = 1/\log n$, this number is at most $\kappa + o(\kappa)$ and the overall number of queries remains $\widetilde{O}(n^{3/2})$, as required. ◀

7 Lower Bounds

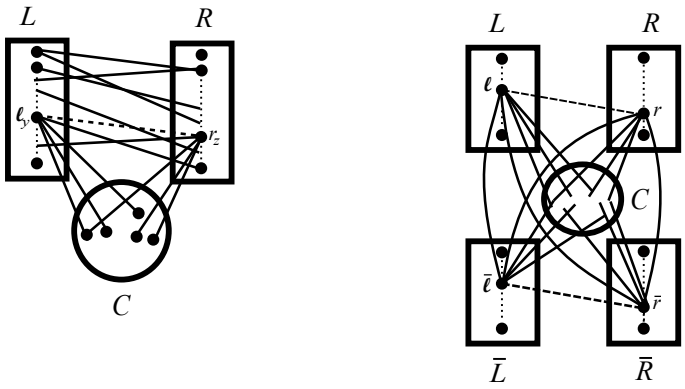
Can we improve the guarantees of our algorithms so that they use at most $\kappa + 1$ colors, rather than $\kappa + o(\kappa)$? After all, every graph G does have a proper $(\kappa(G) + 1)$-coloring. Our lower bounds answer this with a strong "No" in the data streaming and query models. If we insist on a coloring that good, we would incur the worst possible space or query complexity: $\Omega(n^2)$. In fact, this holds even if κ is known to the algorithm in advance. Moreover, all our streaming lower bounds hold even if the input stream consists of edge insertions alone.

Our lower bounds generalize to the problem of producing a $(\kappa + \lambda)$-coloring. We show that this requires $\Omega(n^2/\lambda^2)$ space or query complexity. Such generalizations are based on the following Blow-Up Lemma.

▶ **Definition 15.** *Let G be a graph and λ a positive integer. The* blow-up graph G^λ *is obtained by replacing each vertex of G with a copy of the complete graph K_λ and replacing each edge of G with a complete bipartite graph between the copies of K_λ at its endpoints. More succinctly, G^λ is the lexicographical product $G[K_\lambda]$.*

▶ **Lemma 16** (Blow-Up Lemma)**.** *For all graphs G and positive integers λ, c, if G has a c-clique, then G^λ has a $(c\lambda)$-clique. Also, $\kappa(G^\lambda) \leqslant (\kappa(G) + 1)\lambda - 1$.*

Proof. The claim about cliques is immediate. The bound on $\kappa(G^\lambda)$ follows by taking a degeneracy ordering of G and replacing each vertex v by a list of vertices of the clique that replaces v in G^λ, ordering vertices within the clique arbitrarily. ◀



(a) Gadget graph for Lemma 19. **(b)** Gadget graph for Theorem 21.

Figure 1 Gadget constructions for lower bounds.

Streaming Lower Bounds. Our streaming lower bounds use reductions from the INDEX problem in communication complexity. In the INDEX_N problem, Alice is given a vector $\mathbf{x} = (x_1, \ldots, x_N) \in \{0,1\}^N$ and Bob is given an index $k \in [N]$. The goal is for Alice to send Bob a (possibly random) c-bit message that enables Bob to output x_k with probability at least $2/3$. The smallest c for which such a protocol exists is called the one-way randomized communication complexity, $\text{R}^{\rightarrow}(\text{INDEX}_N)$. As is well known, $\text{R}^{\rightarrow}(\text{INDEX}_N) = \Omega(N)$ [2].

We shall in fact consider instances of INDEX_N where $N = p^2$, for an integer p. Using a canonical bijection between $[N]$ and $[p] \times [p]$, we reinterpret $\mathbf{x}$ as a matrix with entries $(x_{ij})_{i,j\in[p]}$, and Bob's input as $(y, z) \in [p] \times [p]$. We further interpret this matrix $\mathbf{x}$ as the bipartite adjacency matrix of a $(2p)$-vertex balanced bipartite graph $H_{\mathbf{x}}$. Such graphs $H_{\mathbf{x}}$ will be key gadgets in the reductions to follow.

▶ **Definition 17.** *For $\mathbf{x} \in \{0,1\}^{p\times p}$, a realization of $H_{\mathbf{x}}$ on a list $(\ell_1, \ldots, \ell_p, r_1, \ldots, r_p)$ of distinct vertices is a graph on these vertices whose edge set is $\{\{\ell_i, r_j\} : x_{ij} = 1\}$.*

First Flavor: Degeneracy Not Known in Advance. To prove lower bounds of the first flavor, we start by demonstrating the hardness of the abstract problem GRAPH-DIST.

▶ **Definition 18** (GRAPH-DIST problem)**.** *Consider two graph families: $\mathcal{G}_1 := \mathcal{G}_1(n, q, \lambda)$, consisting of n-vertex graphs with chromatic number $\chi \geqslant (q+1)\lambda$, and $\mathcal{G}_2 := \mathcal{G}_2(n, q, \lambda)$, consisting of n-vertex graphs with $\kappa \leqslant q\lambda - 1$. Then GRAPH-DIST$(n, q, \lambda)$ is the problem of distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$ (note that $\mathcal{G}_1 \cap \mathcal{G}_2 = \varnothing$): given an input graph G on n vertices, the problem is to decide whether $G \in \mathcal{G}_1$ or $G \in \mathcal{G}_2$, with success probability at least $2/3$.*

We shall prove that GRAPH-DIST is "hard" in the insertion-only streaming setting and in the query setting, thereby establishing that in these models it is hard to produce a $(\kappa+\lambda)$-coloring. In fact, our proofs will show that it is just as hard to estimate the parameter κ; this goes to show that the hardness of the coloring problem is not just because of the large output size.

▶ **Lemma 19.** *Solving GRAPH-DIST(n, q, λ) in one pass requires $\Omega(n^2/\lambda^2)$ space. More precisely, there is a constant $c > 0$ such that for every integer $\lambda \geqslant 1$ and every sufficiently large integer q, there is a setting $n = n(q, \lambda)$ for which every randomized one-pass streaming algorithm for GRAPH-DIST(n, q, λ) requires at least cn^2/λ^2 bits of space.*

Proof. Put $p = q - 1$. We reduce from INDEX_N, where $N = p^2$, using the following plan. Starting with an empty graph on $n = 3\lambda p$ vertices, Alice adds certain edges based on her input $\mathbf{x} \in \{0,1\}^{p\times p}$ and then Bob adds certain other edges based on his input

$(y, z) \in [p] \times [p]$. By design, solving GRAPH-DIST(n, q, λ) on the resulting final graph reveals the bit x_{yz}, implying that a one-pass streaming algorithm for GRAPH-DIST requires at least $\mathrm{R}^{\rightarrow}(\text{INDEX}_N) = \Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$ bits of memory. The details follow.

First, consider $\lambda = 1$. We use the vertex set $L \uplus R \uplus C$ ("$\uplus$" denotes a disjoint union), where $L = \{\ell_1, \ldots, \ell_p\}$, $R = \{r_1, \ldots, r_p\}$, and $|C| = p$. Alice introduces the edges of the gadget graph $H_{\mathbf{x}}$ (from def. 17), realized on the vertices $(\ell_1, \ldots, \ell_p, r_1, \ldots, r_p)$. Bob introduces all possible edges within $C \cup \{\ell_y, r_z\}$, except for $\{\ell_y, r_z\}$. Let G be the resulting graph (see Figure 1a).

If $x_{yz} = 1$, then G contains a clique on $C \cup \{\ell_y, r_z\}$, whence $\chi(G) \geqslant p + 2$. If, on the other hand, $x_{yz} = 0$, then we claim that $\kappa(G) \leqslant p$. By Lemma 5, the claim will follow if we exhibit a vertex ordering $\lhd$ such that $\text{odeg}_{G,\lhd}(v) \leqslant p$ for all $v \in V(G)$. We use an ordering where $L \cup R \setminus \{\ell_y, r_z\} \lhd \ell_y \lhd \{r_z\} \cup C$ and the ordering within each set is arbitrary. By construction of $H_{\mathbf{x}}$, each vertex in $L \cup R \setminus \{\ell_y, r_z\}$ has *total* degree at most p. For each vertex $v \in \{r_z\} \cup C$, we trivially have $\text{odeg}_{G,\lhd}(v) \leqslant p$ because $|C| = p$. Finally, since $x_{yz} = 0$, the vertex r_z is not a neighbor of ℓ_y; so $\text{odeg}_{G,\lhd}(\ell_y) = |C| = p$. This proves the claim.

When $\lambda \geqslant 1$, Alice and Bob introduce edges so as to create the blow-up graph G^λ, as in Definition 15. By Lemma 16, if $x_{yz} = 1$, then G^λ has a $(p+2)\lambda$-clique, whereas if $x_{yz} = 0$, then $\kappa(G^\lambda) \leqslant (p+1)\lambda - 1$. In the former case, $\chi(G^\lambda) \geqslant (p+2)\lambda = (q+1)\lambda$, so that $G^\lambda \in \mathcal{G}_1(n, q, \lambda)$; cf. Definition 18. In the latter case, $\kappa(G^\lambda) \leqslant q\lambda - 1$, so that $G^\lambda \in \mathcal{G}_2(n, q, \lambda)$. Thus, solving GRAPH-DIST(n, q, λ) on G^λ reveals x_{yz}. ◀

Our coloring lower bounds are straightforward consequences of the above lemma.

▶ **Theorem 20.** *Given a single randomized pass over a stream of edges of an n-vertex graph G, succeeding with probability at least $2/3$ at either of the following tasks requires $\Omega(n^2/\lambda^2)$ space, where $\lambda \geqslant 1$ is an integer parameter:*

- **(i)** *produce a proper $(\kappa + \lambda)$-coloring of G;*
- **(ii)** *produce an estimate $\hat{\kappa}$ such that $|\hat{\kappa} - \kappa| \leqslant \lambda$.*

Furthermore, if we require $\lambda = O\big(\kappa^{\frac{1}{2}-\gamma}\big)$, where $\gamma > 0$, then neither task admits a semi-streaming algorithm.

Proof. An algorithm for either task i and or task ii immediately solves GRAPH-DIST with appropriate parameters, implying the $\Omega(n^2/\lambda^2)$ bounds, thanks to Lemma 19. For the "furthermore" statement, note that the graphs in the family $\mathcal{G}_2$ constructed in the proof of Lemma 19 have $\kappa = \Theta(n)$, so performing either task with the stated guarantee on λ would require $\Omega(n^{1+2\gamma})$ space, which is not in $\widetilde{O}(n)$. ◀

▶ Remark. Together, Theorem 10 and Theorem 20 say that producing a $(\kappa + \kappa/\log^{O(1)} n)$-coloring is possible in semi-streaming space whereas producing a $(\kappa + O\big(\kappa^{\frac{1}{2}-O(1)}\big))$-coloring is not. We leave open the question of whether this gap can be tightened.

Second Flavor: Degeneracy Known in Advance. We now show that the coloring problem remains just as hard even if the algorithm knows the degeneracy of the graph before seeing the edge stream.

▶ **Theorem 21.** *Given as input an integer κ, followed by a stream of edges of an n-vertex graph G with degeneracy κ, a randomized one-pass algorithm that produces a proper $(\kappa + \lambda)$-coloring of G requires $\Omega(n^2/\lambda^2)$ bits of space. Furthermore, if we require $\lambda = O\big(\kappa^{\frac{1}{2}-\gamma}\big)$, where $\gamma > 0$, then the task does not admit a semi-streaming algorithm.*

Proof. We reduce from INDEX_N, where $N = p^2$, using a plan analogous to the one used in proving Lemma 19. Alice and Bob will construct a graph on $n = 5\lambda p$ vertices, using their respective inputs $\mathbf{x} \in \{0,1\}^{p \times p}$ and $(y, z) \in [p] \times [p]$.

First, we consider the case $\lambda = 1$. We use the vertex set $L \uplus R \uplus \overline{L} \uplus \overline{R} \uplus C$, where $L = \{\ell_1, \ldots \ell_p\}$, $R = \{r_1, \ldots, r_p\}$, $\overline{L} = \{\overline{\ell}_1, \ldots, \overline{\ell}_p\}$, $\overline{R} = \{\overline{r}_1, \ldots, \overline{r}_p\}$, and $|C| = p$. Let $\overline{\mathbf{x}}$ be the bitwise complement of $\mathbf{x}$. Alice introduces the edges of the gadget graph $H_{\mathbf{x}}$ (from Definition 17), realized on $L \cup R$, and the edges of $H_{\overline{\mathbf{x}}}$ realized on $\overline{L} \cup \overline{R}$. For ease of notation, put $\ell := \ell_y$, $r := r_z$, $\overline{\ell} := \overline{\ell}_y$, $\overline{r} := \overline{r}_z$, and $S := C \cup \{\ell, r, \overline{\ell}, \overline{r}\}$. Bob introduces all possible edges within S, except for $\{\ell, r\}$ and $\{\overline{\ell}, \overline{r}\}$. Let G be the resulting graph (see Figure 1b).

We claim that the degeneracy $\kappa(G) = p + 2$. To prove this, we consider the case $x_{yz} = 1$ (the other case, $x_{yz} = 0$, is symmetric). By construction, G contains a clique on the $p + 3$ vertices in $C \cup \{\ell, r, \overline{\ell}\}$; therefore, by definition of degeneracy, $\kappa(G) \geqslant p + 2$. To show that $\kappa(G) \leqslant p + 2$, it will suffice to exhibit a vertex ordering $\lhd$ such that $\text{odeg}_{G,\lhd}(v) \leqslant p + 2$ for all $v \in V(G)$. To this end, consider an ordering where $V(G) \setminus S \lhd \overline{\ell} \lhd S \setminus \{\overline{\ell}\}$ and the ordering within each set is arbitrary. Each vertex $v \in V(G) \setminus S$ has $\text{odeg}_{G,\lhd}(v) \leqslant \deg(v) \leqslant p$ and each vertex $v \in S \setminus \{\overline{\ell}\}$ has $\text{odeg}_{G,\lhd}(v) \leqslant \left|S \setminus \{\overline{\ell}\}\right| - 1 = p + 2$. As for the vertex $\overline{\ell}$, since $\overline{x}_{yz} = 1 - x_{yz} = 0$, by the construction in Definition 17, $\overline{r}$ is not a neighbor of $\overline{\ell}$; therefore, $\text{odeg}_{G,\lhd}(\overline{\ell}) \leqslant \left|S \setminus \{\overline{\ell}, \overline{r}\}\right| = p + 2$.

Let $\mathcal{A}$ be a streaming algorithm that behaves as in the theorem statement. Recall that we are considering $\lambda = 1$. Since $\kappa(G) = p + 2$ for every instance of INDEX_N, Alice and Bob can simulate $\mathcal{A}$ on their constructed graph G by first feeding it the number $p + 2$, then Alice's edges, and then Bob's. When $\mathcal{A}$ succeeds, the coloring it outputs is a proper $(p+3)$-coloring; therefore it must repeat a color inside S, as $|S| = p + 4$. But S has exactly one pair of non-adjacent vertices: the pair $\{\ell, r\}$ if $x_{yz} = 0$, and the pair $\{\overline{\ell}, \overline{r}\}$ if $x_{yz} = 1$. Thus, an examination of which two vertices in S receive the same color reveals x_{yz}, solving the INDEX_N instance. It follows that $\mathcal{A}$ must use at least $\text{R}^{\rightarrow}(\text{INDEX}_N) = \Omega(N) = \Omega(p^2)$ bits of space.

Now consider an arbitrary λ. Alice and Bob proceed as above, except that they simulate $\mathcal{A}$ on the blow-up graph G^λ. Since G always has a $(p + 3)$-clique and $\kappa(G) = p + 2$, the two halves of Lemma 16 together imply $\kappa(G^\lambda) = (p + 3)\lambda - 1$. So, when $\mathcal{A}$ succeeds, it properly colors G^λ using at most $(p + 4)\lambda - 1$ colors. For each $A \subseteq V(G)$, abusing notation, let A^λ denote its corresponding set of vertices in G^λ (cf. Definition 15). Since $|S^\lambda| = (p + 4)\lambda$, there must be a color repetition within S^λ. Reasoning as above, this repetition must occur within $\{\ell, r\}^\lambda$ when $x_{yz} = 0$ and within $\{\overline{\ell}, \overline{r}\}^\lambda$ when $x_{yz} = 1$. Therefore, Bob can examine the coloring to solve INDEX_N, showing that $\mathcal{A}$ must use $\Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$ space.

The "furthermore" part follows by observing that $\kappa(G^\lambda) = \Theta\big(|V(G^\lambda)|\big)$. ◀

Query Lower Bounds. We prove lower bounds of the above two flavors for the graph query model as well. We describe the proofs in the full version of the paper [18].

▶ **Theorem 22.** *Given query access to an n-vertex graph G, succeeding with probability at least $2/3$ at either of the following tasks requires $\Omega(n^2/\lambda^2)$ queries, where $\lambda \geqslant 1$ is an integer:*

- **(i)** *produce a proper $(\kappa + \lambda)$-coloring of G;*
- **(ii)** *produce an estimate $\hat{\kappa}$ such that $|\hat{\kappa} - \kappa| \leqslant \lambda$.*

▶ **Theorem 23.** *Given an integer κ and query access to an n-vertex graph G with $\kappa(G) = \kappa$, an algorithm that, with probability $2/3$, produces a proper $(\kappa + \lambda)$-coloring of G must make $\Omega(n^2/\lambda^2)$ queries.*

Combinatorial Lower Bound. Finally, we prove a combinatorial result that shows that unlike $(\Delta + 1)$-coloring, an analogous Palette Sparsification Theorem (as in Assadi et al. [5]) doesn't exist for degeneracy-based coloring. Moreover, our result implies that an algorithm based on such a technique must use at least as many colors as Algorithm 1. We discuss this in detail in the full version of our paper [18].

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. arXiv:1901.01630.

2 Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 175(2):139–159, 1996.

3 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

4 Sabeur Aridhi, Martin Brugnara, Alberto Montresor, and Yannis Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016*, pages 161–168, 2016.

5 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta+ 1)$ vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, page To Appear, 2019.

6 Gary D. Bader and Christopher WV Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, January 2003.

7 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *International Conference on Very Large Data Bases*, 5(5):454–465, 2012.

8 Balabhaskar Balasundaram and Sergiy Butenko. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*, pages 865–890. Springer, 2006.

9 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019.

10 Leonid Barenboim. Deterministic $(\Delta+ 1)$-coloring in sublinear (in Δ) time in static, dynamic, and faulty networks. *Journal of the ACM (JACM)*, 63(5):47, 2016.

11 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

12 Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM (JACM)*, 58(5):23, 2011.

13 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

14 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):20, 2016.

15 Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. *Annals of operations research*, 130(1-4):163–178, 2004.

16 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):40, 2017.

17 Soheil Behnezhad, Mahsa Derakhshan, and Mohammad Taghi Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *CoRR*, abs/1802.10297, 2018.

18 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. *CoRR*, abs/1905.00566, 2019. arXiv:1905.00566.

19 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proc. 39th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20, 2018.

20 K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

21 Gregory J Chaitin. Register allocation & spilling via graph coloring. In *ACM Sigplan Notices*, volume 17, pages 98–105, 1982.

22 Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein. Register allocation via coloring. *Computer languages*, 6(1):47–57, 1981.

23 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\delta+1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 471–480, 2019.

24 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed $(\Delta+ 1)$-coloring algorithm. In *Proc. 50th Annual ACM Symposium on the Theory of Computing*, pages 445–456, 2018.

25 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

26 Fred C Chow and John L Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

27 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

28 P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17(1):61–99, March 1966.

29 Hossein Esfandiari, Silvio Lattanzi, and Vahab S. Mirrokni. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1396–1405, 2018.

30 Martín Farach-Colton and Meng-Tsung Tsai. Tight approximations of degeneracy in large graphs. In *LATIN 2016: Theoretical Informatics*, pages 429–440. Springer Berlin Heidelberg, 2016.

31 Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. In *Annual IEEE Conference on Computational Complexity*, page 278, 1996.

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. Preliminary version in *Proc. 31st International Colloquium on Automata, Languages and Programming*, pages 531–543, 2004.

33 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science*, pages 625–634, 2016.

34 Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

35 Mohsen Ghaffari and Christiana Lymouri. Simple and near-optimal distributed coloring for sparse graphs. In *31st International Symposium on Distributed Computing (DISC 2017)*, page 20, 2017.

36 Mohsen Ghaffari and Ali Sayyadi. Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 142:1–142:14, 2019.

37 Anna C. Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

38 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

39 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.

40 David G Harris, Johannes Schneider, and Hsin-Hao Su. Distributed ($\Delta+$ 1)-coloring in sublogarithmic rounds. In *Proc. 48th Annual ACM Symposium on the Theory of Computing*, pages 465–478, 2016.

41 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 43–52, 2018.

42 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. `arXiv:2002.10142`.

43 Monika Henzinger and Pan Peng. Constant-time dynamic (Δ+1)-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 53:1–53:18, 2020.

44 Öjvind Johansson. Simple distributed $\Delta+$ 1-coloring of graphs. *Information Processing Letters*, 70(5):229–232, 1999.

45 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *International Colloquium on Automata, Languages and Programming*, pages 226–237, 2006.

46 L. Kirousis and D. Thilikos. The linkage of a graph. *SIAM Journal on Computing*, 25(3):626–647, 1996.

47 Kishore Kothapalli and Sriram Pemmaraju. Distributed graph coloring in a few rounds. In *Proc. 30th ACM Symposium on Principles of Distributed Computing*, pages 31–40, 2011.

48 Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489–506, 1979.

49 Vahid Lotfi and Sanjiv Sarin. A graph coloring algorithm for large scale scheduling problems. *Computers & operations research*, 13(1):27–32, 1986.

50 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.

51 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

52 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu. Densest subgraph in dynamic graph streams. In *International Symposium on Mathematical Foundations of Computer Science*, pages 472–482, 2015.

53 Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, 2015.

54 Farnaz Moradi, Tomas Olovsson, and Philippas Tsigas. A local seed selection algorithm for overlapping community detection. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 1–8, 2014.

55 Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

56 Taehoon Park and Chae Y Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research society of Japan*, 39(2):258–265, 1996.

57 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

58 Merav Parter. (Δ+1) coloring in the congested clique model. In *Proc. 45th International Colloquium on Automata, Languages and Programming*, pages 160:1–160:14, 2018.

59 Merav Parter and Hsin-Hao Su. Randomized (Delta+1)-Coloring in O(log* Delta) Congested Clique Rounds. In *Proc. 32nd International Symposium on Distributed Computing*, pages 39:1–39:18, 2018.

60 Jaikumar Radhakrishnan, Saswata Shannigrahi, and Rakesh Venkat. Hypergraph two-coloring in the streaming model. *arXiv preprint*, 2015. `arXiv:1512.04188`.

61 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. URL: `http://networkrepository.com`.

62 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proc. 52nd Annual ACM Symposium on the Theory of Computing*, 2020. `arXiv:1907.10937`.

63 Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 257–266, 2010.

64 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 72:1–72:16, 2018.

65 Michael Stiebitz and Bjarne Toft. A Brooks type theorem for the maximum local edge connectivity. *Electronic Journal of Combinatorics*, 25, March 2016.

66 George Szekeres and Herbert S. Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.

67 Simon Thevenin, Nicolas Zufferey, and Jean-Yves Potvin. Graph multi-coloring for a job scheduling application. *Discrete Applied Mathematics*, 234:218–235, 2018.

68 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. 38th Annual ACM Symposium on the Theory of Computing*, pages 681–690, 2006.