Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring*

Sepehr Assadi† Yu Chen† Sanjeev Khanna†

Abstract

Any graph with maximum degree Δ admits a proper vertex coloring with $\Delta+1$ colors that can be found via a simple sequential greedy algorithm in linear time and space. But can one find such a coloring via a *sublinear* algorithm?

We answer this fundamental question in the affirmative for several canonical classes of sublinear algorithms including graph streaming, sublinear time, and massively parallel computation (MPC) algorithms. In particular, we design:

- A *single-pass* semi-streaming algorithm in dynamic streams using $\widetilde{O}(n)$ space. The only known semi-streaming algorithm prior to our work was a folklore $O(\log n)$-pass algorithm obtained by simulating classical distributed algorithms in the streaming model.
- A sublinear-time algorithm in the standard query model that allows neighbor queries and pair queries using $\widetilde{O}(n\sqrt{n})$ time. We further show that any algorithm that outputs a valid coloring with sufficiently large constant probability requires $\Omega(n\sqrt{n})$ time. No non-trivial sublinear time algorithms were known prior to our work.
- A parallel algorithm in the massively parallel computation (MPC) model using $\widetilde{O}(n)$ memory per machine and $O(1)$ MPC rounds. Our number of rounds significantly improves upon the recent $O(\log\log\Delta \cdot \log^*(n))$-round algorithm of Parter [ICALP 2018].

At the core of our results is a remarkably simple meta-algorithm for the $(\Delta + 1)$ coloring problem: Sample $O(\log n)$ colors for each vertex independently and uniformly at random from the $\Delta + 1$ colors; find a proper coloring of the graph using only the sampled colors of each vertex. As our main result, we prove that the sampled set of colors with high probability contains a proper coloring of the input graph. The sublinear algorithms are then obtained by designing efficient algorithms for finding a proper coloring of the graph from the sampled colors in each model.

We note that all our upper bound results for $(\Delta + 1)$ coloring are either optimal or close to best possible in each model studied. We also establish new lower bounds that rule out the possibility of achieving similar results in these models for the closely related problems of maximal independent set and maximal matching. Collectively, our results highlight a sharp contrast between the complexity of $(\Delta+1)$ coloring vs maximal independent set and maximal matching in various models of sublinear computation even though all three problems are solvable by a simple greedy algorithm in the classical setting.

1 Introduction

Graph coloring is a central problem in graph theory and has numerous applications in diverse areas of computer science. A proper c-coloring of a graph $G(V, E)$ assigns a color to every vertex from the palette of colors $\{1, \ldots, c\}$ such that no edge is monochromatic, i.e., has the same color on both endpoints. An important and well-studied case of graph coloring problems is the $(\Delta + 1)$ coloring problem where Δ is the maximum degree of the graph. Not only does every graph admit a $(\Delta + 1)$ coloring, remarkably, *any* partial coloring of vertices of a graph can be extended to a proper $(\Delta + 1)$ coloring of all vertices: simply pick uncolored vertices in any order and assign a color to a vertex not yet assigned to any of its neighbors; since the max-degree is Δ, such a color always exists.

In this paper, we study the $(\Delta+1)$ coloring problem in the context of processing *massive* graphs. The aforementioned property of $(\Delta+1)$ coloring problem immediately implies a simple (sequential) greedy algorithm for this problem in linear time and space. However, when processing massive graphs, even this algorithm can be computationally prohibitive. This is due to various limitations arising in processing massive graphs such as requiring to process the graph in a streaming fashion on a single machine or in parallel across multiple machines due to storage limita-

*A full version of the paper is available on arXiv [9].

†Department of Computer and Information Science, University of Pennsylvania. Supported in part by the National Science Foundation grant CCF-1617851.
Email: {sassadi,chenyu2,sanjeev}@cis.upenn.edu.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

tions, or simply not having enough time for reading the whole input. A natural question is then:

> *Can we design* **sublinear algorithms** *for* $(\Delta+1)$ *coloring problem in modern models of computation for processing massive graphs?*

We answer this fundamental question in the affirmative for several canonical classes of sublinear algorithms including (dynamic) graph streaming algorithms, sublinear time/query algorithms, and massively parallel computation (MPC) algorithms. We also prove new lower bounds to contrast the complexity of the $(\Delta+1)$ coloring problem in these models with two other closely related Locally Checkable Labeling (LCL) problems (see [37]), namely, the maximal independent set and the maximal matching[1].

1.1 Our Contributions and Techniques We present new sublinear algorithms for the $(\Delta+1)$ coloring problem which are either the first non-trivial ones or significantly improve the state-of-the-art. At the core of these algorithms is a "meta-algorithm" for this problem that we design in this paper; the sublinear algorithms are then obtained by efficiently implementing this meta-algorithm in each model separately.

A Meta-Algorithm for $(\Delta+1)$ *Coloring.* The main approach behind our meta-algorithm is to "sparsify" the $(\Delta+1)$ coloring problem to a list-coloring problem with lists/palletes of size $O(\log n)$ for every vertex. This may sound counterintuitive: while every graph admits a $(\Delta+1)$ coloring that can be found via a simple algorithm, there is no guarantee that it also admits a list-coloring with $O(\log n)$-size lists, let alone one that can be found via a sublinear algorithm. The following key structural result that we prove in this paper, however paves the path for this sparsification.

RESULT 1. (Palette-Sparsification Theorem) *Let $G(V,E)$ be an n-vertex graph with maximum degree Δ. Suppose for any vertex $v \in V$, we sample $O(\log n)$ colors $L(v)$ from $\{1,\ldots,\Delta+1\}$ independently and uniformly at random. Then with high probability there exists a proper $(\Delta+1)$ coloring of G in which the color for every vertex v is chosen from $L(v)$.*

[1] Another closely related LCL problem is the $(2\Delta-1)$ *edge* coloring problem. However, as the output in the edge-coloring problem is linear in the input size, one trivially cannot hope to achieve non-trivial algorithms for this problem in models such as streaming or sublinear time algorithms, and hence we ignore this problem in this paper.

In Result 1, as well as throughout the paper, "with high probability" means with probability $1-1/\text{poly}(n)$ for some large polynomial in n.

Result 1 can be seen as a "sparsification" result for $(\Delta+1)$ coloring: after sampling $O(\log n)$ colors for each vertex randomly, the total number of monochromatic edges is only $O(n\cdot\log^2(n))$ with high probability (see Section 4 for a simple proof); at the same time, by computing a proper coloring of G using only these $O(n\cdot\log^2(n))$ edges—which is promised to exist by Result 1—we obtain a $(\Delta+1)$ coloring of G. As such, Result 1 provides a way of sparsifying the graph into only $\widetilde{O}(n)$ edges, while still allowing for recovery of a $(\Delta+1)$ coloring of the original graph. This sparsification serves as the central tool in our sublinear algorithms for the $(\Delta+1)$ coloring problem.

We shall remark that, as stated, Result 1 only promise the existence of a coloring (which can trivially be found in exponential time), but in fact we show that there is a fast, simple procedure to find the corresponding coloring in linear time, and that this will also be used by our algorithms in each model.

Streaming Algorithms. Our Result 1 can be used to design a *single-pass* semi-streaming algorithm for the $(\Delta+1)$ coloring problem in the most general setting of graph streams, namely, *dynamic* streams that allow both insertions and deletions of edges (see Section 4.1 for details).

RESULT 2. *There exists a randomized single-pass dynamic streaming algorithm for the $(\Delta+1)$ coloring problem using $\widetilde{O}(n)$ space.*

To our knowledge, the only previous semi-streaming algorithm for $(\Delta+1)$ coloring was the folklore $O(\log n)$-pass streaming simulation of the classical $O(\log n)$-round distributed/parallel (PRAM) algorithms for this problem (see, e.g. [35]). No $o(n^2)$ space single-pass streaming algorithm was known for this problem even in insertion-only streams. This state-of-affairs was in fact similar to the case of the closely related maximal matching problem: the best known semi-streaming algorithm for this problem on dynamic streams uses $\Theta(\log n)$ passes [3,33] and it is provably impossible to solve this problem using $o(n^2)$-space in a single pass over a dynamic stream [10] (although this problem is trivial in insertion-only streams). Considering this one might have guessed a similar lower bound also holds for the $(\Delta+1)$ coloring problem. We further prove a lower bound of $\Omega(n^2)$-space on the space complexity of single-pass streaming algorithms for computing a maximal independent set even in insertion-only streams (see Theorem 5.1). Result 2 is in sharp contrast to these results, and

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

shows that $(\Delta+1)$ coloring is provably simpler than both problems in the streaming setting.

Sublinear Time Algorithms. There exists a straightforward greedy algorithm that computes a $(\Delta+1)$ coloring of any given graph in linear time, i.e., $O(m+n)$ time. Perhaps surprisingly, we show that one can improve upon the running time of this textbook algorithm by using Result 1.

RESULT 3. *There exists a randomized $\widetilde{O}(n\sqrt{n})$ time algorithm for the $(\Delta+1)$ coloring problem. Furthermore, any algorithm for this problem requires $\Omega(n\sqrt{n})$ time.*

When designing sublinear (in m) time algorithms, specifying the exact data model is important as the algorithm cannot even read the entire input once. In Result 3, we assume the standard query model for sublinear time algorithms on general graphs (see, e.g., Chapter 10 of Goldreich's book [21]) which allow for two types of queries (i) what is the i-th neighbor of a given vertex v, and (ii) whether a given pair of vertices (u,v) are neighbor to each other or not (see Section 4.2 for details). To our knowledge, this is the first sublinear time algorithm for the $(\Delta+1)$ coloring problem. We also note that an important feature of our algorithm in Result 3 is that it is *non-adaptive*, i.e., it chooses all the queries to the graph beforehand and thus queries are done in parallel.

In yet another contrast to the $(\Delta+1)$ coloring problem, we show that the problem of computing a maximal matching requires $\Omega(n^2)$ queries to the graph and hence $\Omega(n^2)$ time (see Theorem 5.2).

Massively Parallel Computation Algorithms. Another application of our Result 1 is a *constant-round* algorithm for the $(\Delta+1)$ coloring problem in the MPC model, which is a common abstraction of MapReduce-style computation frameworks (see Section 4.3 for formal definition).

RESULT 4. *There exists a randomized MPC algorithm for the $(\Delta+1)$ coloring problem in $O(1)$ MPC rounds on machines with memory $\widetilde{O}(n)$.*

Two recent works considered graph coloring problems in the MPC model. Harvey *et al.* [24] designed algorithms that use $n^{1+\Omega(1)}$ memory per machine and find a $(\Delta+o(\Delta))$ coloring of a given graph—an algorithmically easier problem than $(\Delta+1)$ coloring—in $O(1)$ MPC rounds. Furthermore, Parter [40] designed an MPC algorithm that uses $O(n)$ memory per machine and finds a $(\Delta+1)$ coloring in $O(\log\log\Delta\cdot\log^*(n))$ rounds[2]. Our Result 4 improves these results significantly: both the number of used colors as well as per machine memory compared to [24], and round-complexity (with at most polylog(n)-factor more per-machine memory) compared to [40].

Maximal matching and maximal independent set problems have also been studied previously in the MPC model [20, 30, 33]. Currently, the best known algorithms for these problem with $\widetilde{O}(n)$ memory per machine require $O(\log n)$ rounds in case of maximal matching [33] and $O(\log\log n)$ rounds in case of maximal independent set [20, 30]. For the related problems of $O(1)$-approximating the maximum matching and the minimum vertex cover, a recent set of results achieve $O(\log\log n)$-round MPC algorithms with $\widetilde{O}(n)$ memory per machine [7, 8, 15, 20] (these results however do not extend to the maximal matching problem). Our Result 4 hence is the first example that gives a constant round MPC algorithm for one of the "classic four local distributed graph problems" (i.e., maximal independent set, maximal matching, $(\Delta+1)$ vertex coloring, and $(2\Delta-1)$ edge coloring; see, e.g. [11, 18, 39]), even when the memory per machine is as small as $\widetilde{O}(n)$.

Optimality of Our Sublinear Algorithms: Space-complexity of our streaming algorithm in Result 2 and round-complexity of our MPC algorithm in Result 4 are clearly optimal (to within poly-log factors and constant factors, respectively). We further prove that query and time complexity of our sublinear time algorithm in Result 3 are also optimal up to poly-log factors (see Theorem 5.3).

Perspective: Beyond Greedy Algorithms. Many graph problems admit simple greedy algorithms. Starting with Luby's celebrated distributed/parallel algorithm for the maximal independent set problem [34], there have been numerous attempts in adapting these greedy algorithms to different models of computation including the models considered in this paper (see, e.g. [5, 24, 26, 32, 33, 38]). Typically these adaptations require multiple passes/rounds of computation, and this is for the fundamental reason that most greedy algorithms are inherently sequential: they require accessing the input graph in an *adaptive* manner based on decisions made thus far, which, although limited, still results in requiring *multiple passes/rounds* over the input.

Our work on $(\Delta+1)$ coloring bypasses this limitation of greedy algorithms by utilizing a completely

[2]The algorithm of Parter [40] is stated in the Congested-Clique model, but using the well-known connections between this model and the MPC model, see, e.g. [13,20], this algorithm immediately extends to the MPC model.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

different approach, namely, a non-adaptive sparsification of the input graph (Result 1) that in turns lends itself to space, time, and communication-efficient algorithms in a variety of different models. As such, our results can be seen as an evidence that directly adapting greedy algorithms for graph problems may not necessarily be the best choice in these models. We believe that this viewpoint is an important (non-technical) contribution of our paper as it may pave the way for obtaining more efficient algorithms for other fundamental graph problems in these models.

Our Techniques. The main technical ingredient of our paper is Result 1. For intuition, consider two extreme cases: when the underlying graph is very dense, say is a clique on $\Delta+1$ vertices, and when the underlying graph is relatively sparse, say every vertex (except for one) have degree at most $\Delta/2$. Result 1 is easy to prove for either case albeit by using entirely different arguments. For the former case, consider the bipartite graph consisting of vertices in V on one side and set of colors $\{1,\ldots,\Delta+1\}$ on the other side, where each vertex v in the V-side is connected to vertices in $L(v)$ in the color-side. Using standard results from random graphs theory, one can argue that this graph with high probability has a perfect matching, thus implying the list-coloring of G. For the latter case, consider the following simple greedy algorithm: iteratively sample a color for every vertex from the set $\{1,\ldots,\Delta+1\}$ and assign the color to the vertex if it is not chosen by any of its neighbors so far. It is well-known that this algorithm only requires $O(\log n)$ rounds when number of colors is a constant factor larger than the degree (see, e.g., [41]). As such, the set of colors sampled in the list $L(v)$ for vertices $v \in V$ is enough to "simulate" this algorithm in this case.

To prove Result 1 in general, we need to interpolate between these two extreme cases. To do so, we exploit a graph decomposition result of [23] (see also [14]) for the $(\Delta+1)$ coloring problem, that allows for decomposing a graph into "sparse" and "dense" components. The proof for coloring the sparse components then more or less follows by simulating standard distributed algorithms in [16, 41] as discussed above. The main part however is to prove the result for dense components which requires a global and non-greedy argument. Note that in general, we can always reduce the problem of finding a $(\Delta+1)$ coloring to an instance of the assignment problem on the bipartite graph $V \times \{1,\ldots,\Delta+1\}$ discussed above. The difference is that we need to allow some vertices in $\{1,\ldots,\Delta+1\}$ to be assigned to more than one vertex in V when $|V| > \Delta+1$ (as opposed to the case of cliques above that only required finding a perfect matching). We show that if the original graph is "sufficiently close" to being a clique, then with high probability, such an assignment exists in this bipartite graph and use this to prove the existence of the desired list-coloring of G.

Result 1 implies the sublinear algorithms we design in each model with a simple caveat: The list-coloring problem that needs to be solved in the sparsified graph is in general NP-hard and hence using this result directly does not allow for a polynomial time implementation of our algorithms. We thus combine Result 1 with additional ideas specific to each model to turn these algorithms into polynomial time (and in fact even sublinear time) algorithms.

2 Preliminaries

Notation. For any $t \geq 1$, we define $[t] := \{1,\ldots,t\}$. For a graph $G(V,E)$, we use $V(G) := V$ to denote the vertices, $E(G) := E$ to denote the edges, and $\deg(v)$ to denote the degree of $v \in V$.

2.1 The Harris-Schneider-Su Network Decomposition [23]

In the proof of our Result 1, we use a network decomposition result of Harris, Schneider and Su for designing distributed algorithms for graph coloring in the LOCAL model [23]. We emphasize that we use of this decomposition in a quite different way than the ones in distributed settings [14, 23].

The Harris-Schneider-Su network decomposition, henceforth HSS-decomposition, partitions a graph $G(V,E)$ into *sparse* and *dense* regimes, measured with respect to a parameter $\varepsilon \in [0,1)$.

DEFINITION 1. (ε-FRIEND EDGES) *For any $\varepsilon \in [0,1)$, we say that an edge (u,v) in a graph G is an* ε-friend *edge iff $|N(u) \cap N(v)| \geq (1-\varepsilon)\Delta$. Let $F_\varepsilon \subseteq E(G)$ denote the set of all ε-friend edges.*

DEFINITION 2. (ε-DENSE VERTICES) *For any $\varepsilon \in [0,1)$ we say that a vertex v in a graph G is* ε-dense *iff degree of v in F_ε is at least $(1-\varepsilon)\Delta$. We use $V_\varepsilon^{\text{dense}}$ to denote the set of all ε-dense vertices.*

Consider the graph $H_\varepsilon := H(V_\varepsilon^{\text{dense}}, F_\varepsilon)$ as the subgraph of G on the set of ε-dense vertices and containing only the ε-friend edges. Let $C_1,\ldots,C_k$ be the connected components of H_ε. HSS-decomposition partitions the vertices of the graph into $V_\varepsilon^{\text{dense}}$ and $V \setminus V_\varepsilon^{\text{dense}}$, where $V_\varepsilon^{\text{dense}}$ is partitioned into $C_1,\ldots,C_k$ with the following properties given by Lemma 2.1 and Proposition 2.1.

LEMMA 2.1. ([23]) *Any connected component C_i of H_ε has size at most $(1+3\varepsilon)\Delta$. Moreover, any vertex $v \in C_i$ has at most:*

1. *$\varepsilon\Delta$ neighbors (in G) in $V_\varepsilon^{\text{dense}} \setminus C_i$, i.e., $\left|N_G(v) \cap (V_\varepsilon^{\text{dense}} \setminus C_i)\right| \leq \varepsilon\Delta$.*

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

2. $3\varepsilon\Delta$ *non-neighbors (in G) in C_i, i.e.,* $|C_i \setminus N_G(v)| \leq 3\varepsilon\Delta$.

Define *ε-sparse* vertices as $V_\varepsilon^{\text{sparse}} := V \setminus V_\varepsilon^{\text{dense}}$, i.e., the vertices which are not ε-dense. The main property of ε-sparse vertices we are interested in is as follows (proof appears in full version [9]).

PROPOSITION 2.1. *Let v be any ε-sparse vertex in G. Then, the total number of edges spanning the neighborhood of v is at most $(1-\varepsilon^2)\cdot\binom{\Delta}{2}$, that is $|\{(u,w) \mid u \in N_G(v) \wedge w \in N_G(v)\}| \leq (1-\varepsilon^2)\cdot\binom{\Delta}{2}$.*

A Simple Extension of the HSS-Decomposition. It would be more convenient for us to work with a slightly different variant of the HSS-decomposition that we introduce here.

LEMMA 2.2. (EXTENDED HSS-DECOMPOSITION) *For any parameter $\varepsilon \in [0,1)$, any graph $G(V,E)$ can be decomposed into a collection of vertices $V := V_\star^{\text{sparse}} \cup C_1 \cup \ldots \cup C_k$ such that:*

1. *$V_{2\varepsilon}^{\text{sparse}} \subseteq V_\star^{\text{sparse}} \subseteq V_\varepsilon^{\text{sparse}}$, i.e., any vertex in $V_\star^{\text{sparse}}$ is at least (2ε)-sparse and at most ε-sparse.*
2. *For any $i \in [k]$, C_i has the following properties (we refer to C_i as an almost-clique):*
   (a) *$(1-\varepsilon)\Delta \leq |C_i| \leq (1+6\varepsilon)\Delta$.*
   (b) *Any $v \in C_i$ has at most $7\varepsilon\Delta$ neighbors outside of C_i.*
   (c) *Any $v \in C_i$ has at most $6\varepsilon\Delta$ non-neighbors inside of C_i.*

Two main differences between Lemma 2.1 and the original HSS-decomposition are: (i) size of each C_i is now lower bounded (HSS-decomposition does not lower bound the size of C_i), and (ii) the number of all neighbors of any vertex outside C_i is now bounded (not only neighbors to other dense vertices as in the original HSS-decomposition). The proof of this lemma appears in full version [9].

3 The Palette-Sparsification Theorem

We prove our Result 1 in this section.

THEOREM 3.1. (Palette-Sparsification Theorem) *Let $G(V,E)$ be any n-vertex graph and Δ be the maximum degree in G. Suppose for each vertex $v \in V$, we independently pick a set $L(v)$ of colors of size $\Theta(\log n)$ uniformly at random from $[\Delta+1]$. Then with high probability there exists a proper coloring $\mathcal{C} : V \to [\Delta+1]$ of G such that for all vertices $v \in V$, $\mathcal{C}(v) \in L(v)$.*

In the full version of the paper [9], we also show that the bound of $O(\log n)$ on number of sampled colors in this result is optimal up to lower order terms.

Let us start by fixing the parameters used in the proof of Theorem 3.1. Let $\varepsilon > 0$ be a sufficiently small constant, say, $\varepsilon := 1/5000$ and $\alpha > 0$ be a sufficiently large integer, say $\alpha = 5000$[3]. In Theorem 3.1, we make each vertex sample $\left(\alpha \cdot \log n/\varepsilon^2\right)$ colors in $L(v)$. We assume that $(\Delta+1) > \alpha\cdot\log n/\varepsilon^2$; otherwise Theorem 3.1 is trivial as we sampled all $\Delta+1$ colors for each vertex and every graph admits a $(\Delta+1)$ coloring. For the purpose of the analysis, we assume that the set $L(v)$ of each vertex is union of three sets $L_1(v) \cup L_2(v) \cup L_3(v)$, named *batch* one, two, and three, respectively, where each $L_i(v)$ for $i \in [3]$ is created by picking each color in $[\Delta+1]$ independently and with probability $p := \left(\frac{\alpha\cdot\log n}{3\varepsilon^2\cdot(\Delta+1)}\right)$. While this distribution is not identical to the one in Theorem 3.1, it is easy to see that proving the theorem for this distribution also implies the original result as in this new distribution, with high probability, no vertex samples more than $O(\log n)$ colors.

We use the extended HSS-decomposition with parameter ε (Lemma 2.2): graph G is decomposed into $V := V_\star^{\text{sparse}} \cup C_1 \cup \ldots \cup C_k$ where each C_i for $i \in [k]$ is an almost-clique.

We prove Theorem 3.1 in three parts. In the first part, we argue that by only using the colors in the first batch $L_1(\cdot)$, we can color all the vertices in $V_\star^{\text{sparse}}$. This part is mostly standard and more or less follows from the results in [14, 16, 23] by simulating a distributed local algorithm using only the colors in the first batch. We hence concentrate bulk of our effort in proving the next two parts which are the key components of the proof. We first show that using only the colors in the second batch, we can color a relatively large fraction of vertices in each almost-clique C_i *at a rate of two vertices per color* (assuming the number of non-edges in the almost-clique is not too small). This allows us to "save" extra colors for coloring the remainder of the almost-cliques, which we do in the last part. We note that unlike the coloring of the first part which is based on a "local" coloring scheme (in which we determine the color of each vertex based on colors assigned to each of its neighbors similar to the greedy algorithm), the coloring of the second and third part is done in a "global" manner in which the color of a vertex is determined based on some global properties of the

[3] In the interest of simplifying the exposition of the proof, we made no attempt in optimizing the constants. The proof of the theorem can be made to work with much smaller constants than the ones used here.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

graph not only the local neighborhood of a vertex.

Partial Coloring Function. Define a function $\mathcal{C} : V \to [\Delta+1] \cup \{\perp\}$ that assigns one of the colors in $[\Delta+1]$ plus the *null color* $\perp$ to the vertices, such that no two neighboring vertices have the same color from $[\Delta+1]$ (but they may both have the null color $\perp$). We refer to $\mathcal{C}$ as a *partial coloring* function and refer to vertices that are colored by $\mathcal{C}$ in $[\Delta+1]$ as having a *valid color*. Furthermore, we say that a valid color c is *available* to a vertex $v \in V$ in the partial coloring $\mathcal{C}$, iff $\mathcal{C}$ does not assign c to any neighbor of v. The set of available colors for v is denoted by $A_{\mathcal{C}}(v)$.

It is immediate that if $\mathcal{C}$ does not assign a null color to any vertex, then the resulting coloring is a proper $(\Delta+1)$-coloring of the graph. We start with a partial coloring function $\mathcal{C}$ which assigns a null color to all vertices initially and modify this coloring in each part to remove all null colors.

3.1 Part One: Coloring Sparse Vertices. Recall the definition of sparse vertices $V_\star^{\text{sparse}}$ in the extended HSS-decomposition from Section 2. In the first part of the proof, we show that we can color all sparse vertices using only the colors in the first batch.

LEMMA 3.1. *With high probability, there exists a partial coloring function $\mathcal{C}_1 : V \to [\Delta+1] \cup \{\perp\}$ such that for all vertices $v \in V_\star^{\text{sparse}}$, $\mathcal{C}_1(v) \in L_1(v)$.*

As the proof of this lemma closely follows the previous work in [14, 16, 23] with only some minor modifications, we postpone its proof to the full version [9].

3.2 Part Two: Initial Coloring of Almost Cliques. Recall that by Lemma 3.1, after the first part, we have a partial coloring $\mathcal{C}_1 : V \to [\Delta+1] \cup \{\perp\}$ that assigns a valid color to all sparse vertices. We now design a partial coloring $\mathcal{C}_2 : V \to [\Delta+1] \cup \{\perp\}$ where for all $v \in V_\star^{\text{sparse}}$, $\mathcal{C}_2(v) := \mathcal{C}_1(v)$ and $\mathcal{C}_2(v) = \perp$ for remaining vertices initially but some additional vertices would also be assigned a valid color by the end of this part using the second batch.

Fix the almost-cliques $C_1, \ldots, C_k$. Define $\overline{C}_i$ as the *complement-graph* of C_i on the same set of vertices as C_i by switching edges and non-edges in C_i. Note that any two neighboring vertices in $\overline{C}_i$ *can be colored the same* (in G). We exploit this fact in the following definition.

DEFINITION 3. (COLORFUL MATCHING) *We say that a matching M_i in the complement-graph $\overline{C}_i$ of an almost-clique is a* colorful matching *with respect to the partial coloring $\mathcal{C}_2$ iff:*

1. *For any $(u,v) \in M_i$ there is a color $c_{u,v}$ s.t $c_{u,v} \in L_2(u) \cap L_2(v)$ and $c_{u,v} \in A_{\mathcal{C}_2}(u) \cap A_{\mathcal{C}_2}(v)$.*
2. *For any pairs of edges $(u_1,v_1),(u_2,v_2) \in M_i$, $c_{u_1,v_1} \neq c_{u_2,v_2}$.*

By finding a colorful matching in a complement-graph $\overline{C}_i$, we can "save" on the the colors needed for coloring C_i as we can color vertices of the matching at a rate of two vertices per color.

We now iterate over complement-graphs $\overline{C}_1, \ldots, \overline{C}_k$ one by one, and show that there exists a sufficiently large colorful matching in each complement-graph, *even after* we update the coloring $\mathcal{C}_2$ for vertices matched by the colorful matchings in previous complement-graphs.

LEMMA 3.2. *Fix any complement-graph $\overline{C}$ and let $\mathcal{C}_2 : V \to [\Delta+1] \cup \{\perp\}$ be any partial coloring in which $c(v) = \perp$ for all $v \in \overline{C}$. Suppose* average degree *of $\overline{C}$ is $\bar{d}$. Then, there exists a colorful matching of size at least $\left(\bar{d}/(320\varepsilon)\right)$ in $\overline{C}$ with high probability (over the randomness of L_2).*

We start by some definitions. For $(u,v) \in \bar{C}$, a color is *available* to this edge if the color is available to both u and v under $\mathcal{C}_2$. For a set of colors D, let $a_D(e)$ be the number of available colors for an edge e in D. For a set of edges F, we define $a_D(F) := \sum_{e \in F} a_D(e)$. We say that an edge $e = (u,v)$ sampled an available color in L_2 iff there exists an available color c for e in $L_2(u) \cap L_2(v)$. Lemma 3.2 relies on the following lemma.

LEMMA 3.3. *Let $\bar{C}'$ be a subgraph of $\bar{C}$ and $F = E(\bar{C}')$ be its edge-set. Let D be any set of colors such that $a_D(F) \geq 120\epsilon^2\Delta^2$. If for each vertex in $\bar{C}'$, we sample each color in D with probability $p = \frac{20 \log n}{\epsilon\Delta}$, then with high probability, there is an edge (u,v) in F that samples an available color.*

Proof. Let $q := \frac{1}{\varepsilon\Delta}$. We argue that if each vertex samples each color in D with probability q, then with a constant probability, there is an edge (u,v) in F that samples an available color. We then argue that sampling with rate p can be seen as performing this experiment independently $O(\log n)$ times and obtain the final high probability bound.

For an edge $e = (u,v)$, let X_e be an indicator random variable which is 1 iff e sampled an available color (in the experiment with probability q). Since $q^2 \cdot a_D(e) \leq \frac{\Delta}{\epsilon^2\Delta^2} < 1$, we have,

$$\mathbb{E}[X_e] = 1 - \Pr[X_e = 0] = 1 - \left(1-q^2\right)^{a_D(e)}$$
$$(\text{as } 1 - x \leq e^{-x} \leq 1 - x/2 \text{ for } x < 1)$$
$$\geq 1 - \exp\left(-q^2 \cdot a_D(e)\right) \geq \frac{q^2 \cdot a_D(e)}{2}.$$

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Define $X = \sum_{e \in F} X_e$. We prove $\Pr[X > 0] \geq \frac{1}{15}$ which implies that with probability at least $1/15$, an edge in F samples an available color.

Firstly, notice that $\mathbb{E}[X] = \sum_{e \in F} \mathbb{E}[X_e] \geq \frac{a_D(F) \cdot q^2}{2} \geq 120\varepsilon^2 \cdot \Delta^2 \cdot \frac{1}{2\varepsilon^2\Delta^2} = 60$. We prove that the variance of X is not much larger than its expectation, and use Chebyshev's inequality to prove the bound on $\Pr[X > 0]$. By definition, $\mathrm{Var}[X] = \sum_{e \in F} \mathrm{Var}[X_e] + \sum_{e \neq e' \in F} \mathrm{Cov}[X_e, X_{e'}]$. Since each $X_e \in \{0, 1\}$, we have $\mathrm{Var}[X_e] \leq \mathbb{E}[X_e]$, hence it only remains to bound the covariance terms.

For any pair of edges e, e' in F, if they do not share a common endpoint, then the variables X_e and $X_{e'}$ are independent (hence their covariance is 0), but if they share a common endpoint, their covariance would be non-zero. However, in this case, $\mathrm{Cov}[X_e, X_{e'}] \leq \mathbb{E}[X_e \cdot X_{e'}] \leq 1 - \left(1 - q^3\right)^{a_D(e)} \leq 1 - \exp\left(-2q^3 \cdot a_D(e)\right) \leq 2a_D(e) \cdot q^3$. By Property (2c) of Lemma 2.2, each vertex in $\bar{C}$ has at most $6\varepsilon\Delta$ neighbors (as edges in $\bar{C}$ are non-edges in the almost-clique C). As such, there are at most $12\epsilon\Delta$ edges that share a common endpoint with an edge e. Let $S(e)$ denote the set of edges in F that share an endpoint with e. We have,

$$\begin{aligned}\mathrm{Var}[X] &\leq \sum_{e \in F} \mathbb{E}[X_e] + \sum_{e \in F} \sum_{e' \in S(e)} 2a_D(e)q^3 \\ &\leq \mathbb{E}[X] + \sum_{e \in F} a_D(e) \cdot 24\epsilon\Delta q^3 \leq 50\,\mathbb{E}[X].\end{aligned}$$

The last equation is because $q = \frac{1}{\epsilon\Delta}$. By Chebyshev's inequality: $\Pr[X = 0] \leq \frac{\mathrm{Var}[X]}{\mathbb{E}[X]^2} \leq \frac{5}{6}$.

So if we sample each color with probability q, there is an edge (u, v) that samples an available color with probability at least $\frac{1}{6}$. By sampling the colors at rate $p = 20 \log n \cdot q$, we can repeat this trial at least $10 \log n$ times and obtain that with $1 - (1/6)^{10 \log n} \geq 1 - 1/n^3$, there is an edge that sampled an available color. ∎

Proof. [Proof of Lemma 3.2] We construct the colorful matching in the following manner. We iterate over the colors (in an arbitrary order) and for each color c, we find the vertices which sampled this color in $L_2(\cdot)$ (this choice is independent across colors by the sampling process that defines $L_2(\cdot)$). If c is available for some edge (u, v) in $\bar{C}$, we add (u, v) with color $c_{u,v} = c$ to the matching, delete this edge from the graph, and move on to the next color. Clearly the resulting matching will be colorful (as in Definition 3). It thus remains to lower bound the size of this matching.

Let $\bar{C}'$ be $\bar{C}$ initially and F be its edge-set, i.e., $F = E(\bar{C}')$. D is also initially the set of all colors in $[\Delta + 1]$. Let N be the number of vertices in $\bar{C}$. Consider the value of $a_D(F)$ throughout the process. When we are dealing with a color c, if we cannot find an edge $(u, v) \in \bar{C}'$ where c is available for (u, v), we delete the color c from D. In this case, $a_D(F)$ will decrease by $a_c(F) \leq |F|$. Otherwise, we add (u, v) with color c to our colorful matching, delete c from D, and delete u and v from $\bar{C}'$. In this case, $a_D(F)$ will decrease by at most $a_c(F) + 12\varepsilon\Delta^2 \leq \bar{d}\Delta + 12\varepsilon\Delta^2 \leq 18\varepsilon\Delta^2$ since each vertex in $\bar{C}'$ has at most $6\varepsilon\Delta$ neighbors (by Property (2c) of extended HSS-decomposition in Lemma 2.2) and $a_c(F)$ is at most $|F| = \bar{d}N/2 \leq \bar{d} \cdot (1 + 6\varepsilon)\Delta/2 \leq \bar{d}\Delta$ as in the extended HSS-decomposition, $N = \left|\bar{C}\right| \leq (1 + 6\varepsilon)\Delta$ (by Property (2a) of Lemma 2.2). By Lemma 3.3 (as the process of sampling colors in $L_2(\cdot)$ is identical to the lemma statement but sampling colors with higher probability), with high probability, $a_D(F)$ will decrease by at most $120\varepsilon\Delta^2$ before we add a new edge to the colorful matching. So each time when we add a new edge into the colorful matching, $a_D(F)$ decreases by at most $138\varepsilon\Delta^2$ with high probability. We now lower bound the value of $a_D(F)$ which allows us to bound the number of times an edge is added to the colorful matching.

Let $e = (u, v)$ be an edge in F. Both u and v belong to the almost-clique C in the extended HSS-decomposition and hence by Lemma 2.2, each have at most $7\varepsilon\Delta$ neighbors outside C. This suggests that even if $\mathcal{C}_2$ has assigned a color to all vertices except for C, there are at least $(1 - 14\varepsilon)\Delta$ available colors for the edge e, i.e., $a_D(e) \geq (1 - 14\varepsilon)\Delta$. Moreover, by Lemma 2.2, we also have that the number of vertices in the almost-clique C and hence also in $\bar{C}$ is $N \geq (1 - \varepsilon)\Delta$. As such,

$$\begin{aligned}a_D(F) &= \sum_{e \in F} a_D(e) \geq |F|\,(1 - 14\epsilon)\Delta \\ &= (1 - 14\epsilon) \cdot \frac{\bar{d}N}{2} \cdot \Delta \geq \frac{1}{2}(1 - 14\epsilon)(1 - \epsilon)\bar{d}\Delta \\ &\geq 0.45\bar{d}\Delta^2,\end{aligned}$$

by the choice of ε. Consequently, before $a_D(F)$ becomes smaller than $120\varepsilon\Delta^2$ (and we could no longer apply Lemma 3.3), we would have added at least $\frac{0.45\bar{d}\Delta^2}{138\epsilon\Delta^2} \geq \frac{\bar{d}}{320\epsilon}$ edges to the colorful matching with high probability, finalizing the proof. ∎

The coloring $\mathcal{C}_2$ is then computed as follows. We iterate over almost-cliques $C_1, \ldots, C_k$ and for each one, we pick a colorful matching of size $\frac{\bar{d}}{320\varepsilon} \geq 4\bar{d}$ (by our choice of ε); by Lemma 3.2, we find this matching with high probability. We only pick $4\bar{d}$ edges from this colorful matching and for each edge (u, v) in

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

these $4\bar{d}$ edges, we set $\mathcal{C}_2(u) = \mathcal{C}_2(v) = c_{u,v}$. By Definition 3, this is a valid coloring. We then move to the next almost-clique (and use Lemma 3.2 with the updated $\mathcal{C}_2$).

3.3 Part Three: Final Coloring of Almost–Cliques. We now finalize the coloring of almost-cliques and obtain a coloring $\mathcal{C}_3 : V \to [\Delta+1] \cup \{\perp\}$ that assigns a valid color to all vertices. Initially, $\mathcal{C}_3(v) = \mathcal{C}_2(v)$ for all $v \in V$. We then iterate over almost-cliques $C_1, \ldots, C_k$ and update $\mathcal{C}_3$ to assign a valid color to all vertices of the current almost-clique. For any C_i, define $\widehat{C}_i$ as the vertices that are not yet assigned a valid color in $\mathcal{C}_3$.

DEFINITION 4. (PALETTE-GRAPH) *For any almost-clique C_i in G and a partial coloring $\mathcal{C}_3$, we define a* palette-graph *H_i of the almost-clique with respect to $\mathcal{C}_3$ as follows:*

- *H_i is a bipartite graph between the uncolored vertices in C_i (i.e., $\widehat{C}_i$) and colors $[\Delta+1]$.*
- *There exists an edge between $u \in \widehat{C}_i$ and $c \in [\Delta+1]$ iff the color c is available to vertex u according to $\mathcal{C}_3$ (i.e., $c \in A_{\mathcal{C}_3}(u)$) and moreover $c \in L_3(u)$.*

Suppose we are able to find a matching in the palette-graph of an almost-clique C_i that matches all vertices in $\widehat{C}_i$. Let c_v be the matched pair of $v \in \widehat{C}_i$. We set $\mathcal{C}_3(v) = c_v$ and correctly color all vertices in this almost-clique, and then continue to the next almost-clique. The following lemma proves that with high probability, we can find such a matching in every almost-clique.

LEMMA 3.4. *Let C_i be any almost-clique in G and $\mathcal{C}_3$ be the partial coloring obtained after processing almost-cliques $C_1, \ldots, C_{i-1}$. With high probability (over the randomness of the third batch), there exists a matching in the palette-graph of C_i that matches all vertices in $\widehat{C}_i$.*

Proof of this lemma is based on the following result on existence of large matchings in certain random graphs that we prove in this paper.

LEMMA 3.5. *Suppose $N \leq n$ and $0 \leq \delta \leq \frac{1}{12}$. Let $H = (V_1, V_2)$ be a bipartite graph such that:*

1. *$|V_1| \leq (1-3\delta)N$ and $|V_2| \leq 2N$;*
2. *each vertex in V_1 has degree at least $\frac{2N}{3}$ and at most N;*
3. *the average degree of vertices in V_1 is at least $(1-\delta)N$.*

A subgraph of H obtained by sampling each edge with probability $p = \frac{90 \log n}{N}$ has a matching of size $|V_1|$ with high probability.

The proof of Lemma 3.5 is quite technical and hence we postpone it to the full version of the paper. We now use this lemma to prove Lemma 3.4.

Proof. [Proof of Lemma 3.4] Define H_i as the bipartite graph with the same vertex set as the palette-graph of C_i such that there is an edge between a vertex v and a color c iff c is available to v (edges in H_i are superset of the ones in palette-graph as an edge (v,c) can appear in H_i even if $c \notin L_3(v)$). By this definition, the palette-graph of C_i is a subgraph of H_i chosen by picking each edge independently with probability $\frac{\alpha \log n}{3\varepsilon^2\Delta} \geq \frac{100 \log n}{\Delta}$ (by the choice of L_3).

We apply Lemma 3.5 to a properly chosen subgraph $\bar{H}$ of H_i with the same vertex-set to prove this lemma. Let n_i be the number of vertices in C_i. By definition of coloring $\mathcal{C}_2$ (after the proof of Lemma 3.2), $\widehat{C}_i$ has $n_i - 2 \cdot 4\bar{d} = n_i - 8\bar{d}$ vertices. For any vertex $v \in \widehat{C}_i$, if v has more than $n_i - 4\bar{d}$ available colors (i.e., neighbors in H_i), then we pick $n_i - 4\bar{d}$ available colors for v arbitrarily and only connect v to those color-vertices in $\bar{H}$; otherwise, v has the same neighbors in $\bar{H}$ and H_i.

We prove that $\bar{H}$ satisfies all three properties in Lemma 3.5. Let $N = n_i - 4\bar{d}$ and $\delta = \frac{1.3\bar{d}}{N}$, $V_1 = \widehat{C}_i$ and $V_2 := [\Delta+1]$, and thus $|V_1| = \left|\widehat{C}_i\right| = (n_i - 8\bar{d}) < (1-3\delta)N$. This proves the first part of Property (1) of Lemma 3.5. Moreover, as C_i is an almost-clique, $n_i \geq (1-\varepsilon)\Delta$ by Property (2a) and $\bar{d} \leq 6\varepsilon\Delta$ by Property (2c) of Lemma 2.2, and hence $2N = 2(n_i - 4\bar{d}) \geq 2((1-\epsilon)\Delta - 4 \cdot 6\epsilon\Delta) > \Delta + 1 = |V_2|$, proving the second part of Property (2) as well.

Furthermore, each vertex in $\widehat{C}_i$ has degree at most $n_i - 4\bar{d} = N$. Also any vertex in $\widehat{C}_i$ has degree at most $7\varepsilon\Delta$ outside the almost-clique in G by Property (2b) of Lemma 2.2, so any vertex in $\widehat{C}_i$ has at least $(1-7\varepsilon)\Delta - 4\bar{d} > \frac{2N}{3}$ available colors (even if $\mathcal{C}_3$ has assigned colors to all vertices outside C_i and all colors used by the colorful matching are also unavailable). As in $\bar{H}$ we connect every vertex to the available color-vertices, $\bar{H}$ satisfies Property (2) in Lemma 3.5.

Consider a vertex $v \in \widehat{C}_i$. Let $\bar{d}_v$ be the number of non-neighbors of v inside C_i and hence v has at most $\Delta - (n_i - \bar{d}_v - 1)$ neighbors outside C_i. As such, v has at least $n_i - \bar{d}_v - 4\bar{d}$ neighbors inside of H_i ($4\bar{d}$ is the number of colors used by the colorful matching), hence also at least $n_i - \bar{d}_v - 4\bar{d}$ neighbors inside of $\bar{H}$. So $\bar{H}$ has at least $\left|\widehat{C}_i\right|(n_i - 4\bar{d}) - \bar{d}n_i \geq \left|\widehat{C}_i\right|(n_i - 5.2\bar{d})$ edges (by the fact that $\left|\widehat{C}_i\right| = n_i - 8\bar{d} \geq n_i - 48\varepsilon\Delta$

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

and the choice of ε). Hence, the average degree in $\bar{H}$ is at least $n_i - 5.2\bar{d} = N - 1.2\bar{d} \geq (1-\delta)N$, which implies $\bar{H}$ satisfies Property (3) in Lemma 3.5.

To conclude, $\bar{H}$ satisfies the properties of Lemma 3.5. Since the palette-graph of C_i contains a subgraph of $\bar{H}$ obtained by sampling each edge of $\bar{H}$ with probability $\frac{100\log n}{\Delta} \geq \frac{90\log n}{N}$ (as argued above), the palette-graph contains a matching of size $|V_1| = \left|\widehat{C}_i\right|$ with high probability. ∎

3.4 Wrap-Up: Proof of Theorem 3.1. The existence of list-coloring under the sampling process of L_1, L_2 and L_3 follows from Lemmas 3.1, 3.2 and 3.4. Note that this sampling process is not exactly the same as sampling $O(\log n)$ colors uniformly at random as in Theorem 3.1. However, in this process, with high probability, we do not sample more than $O(\log n)$ colors for each vertex and hence conditioning on sampling $O(\log n)$ colors (as in Theorem 3.1) only changes the probability of success by a negligible factor, hence implying Theorem 3.1.

4 Sublinear Algorithms for $(\Delta+1)$ Coloring

We now use our palette-sparsification theorem to design sublinear algorithms for $(\Delta+1)$ coloring in different models of computation. We start by introducing a "meta-algorithm" for $(\Delta+1)$ coloring, called ColoringAlgorithm, and in the subsequent sections show how to implement it in each model.

The Meta-Algorithm. The algorithm is as follows:

ColoringAlgorithm(G, Δ): A meta-algorithm for finding a $(\Delta+1)$-coloring in a graph $G(V,E)$ with maximum degree Δ.

1. Sample $\Theta(\log n)$ colors $L(v)$ uniformly at random for each vertex $v \in V$ (as in Theorem 3.1).
2. Define, for each color $c \in [\Delta+1]$, a set $\chi_c \subseteq V$ where $v \in \chi_c$ iff $c \in L(v)$.
3. Define E_{conflict} as the set of all edges (u,v) where both $u, v \in \chi_c$ for some $c \in [\Delta+1]$.
4. Construct the *conflict graph* $G_{\text{conflict}}(V, E_{\text{conflict}})$.
5. Find a proper list-coloring of $G_{\text{conflict}}(V, E_{\text{conflict}})$ with $L(v)$ being the color list of vertex $v \in V$.

We refer to ColoringAlgorithm as a "meta-algorithm" since constructing the conflict graph as well as finding its list-coloring are *unspecified* steps in ColoringAlgorithm. To implement this meta-algorithm in different models, we need to come up with an efficient way of performing these two tasks which are model-specific and are hence not fixed in ColoringAlgorithm. The following lemma establishes the main properties of ColoringAlgorithm (the proof is an application of Theorem 3.1 together with Chernoff bound and appears in the full version [9]).

LEMMA 4.1. *Let $G(V,E)$ be a graph with maximum degree Δ. In* ColoringAlgorithm(G,Δ), *with high probability:*

1. *The output is a valid $(\Delta+1)$ coloring of the graph G.*
2. *For any $c \in [\Delta+1]$, size of χ_c is $O(n\log n/\Delta)$.*
3. *The maximum degree in graph G_{conflict} is $O(\log^2 n)$.*

4.1 A Single-Pass Streaming Algorithm We first give an application of our palette-sparsification theorem in designing a dynamic streaming algorithm for the $(\Delta+1)$ coloring problem. In the dynamic streaming model, the input graph is presented as an arbitrary sequence of edge insertions and deletions and the goal is to analyze properties of the resulting graph using memory that is sublinear in the input size, which is proportional to the number of edges in the graph. We are particularly interested in algorithms that use $O(n \cdot \text{polylog}(n))$ space, referred to as *semi-streaming* algorithms [17].

THEOREM 4.1. *There exists a randomized single-pass semi-streaming algorithm that given a graph G with maximum degree Δ presented in a dynamic stream, with high probability finds a $(\Delta+1)$ coloring of G with using $\widetilde{O}(n)$ space and polynomial time*[4].

We prove Theorem 4.1 by implementing ColoringAlgorithm in dynamic streams. Recall that implementing ColoringAlgorithm requires us to specify (i) how we construct the conflict-graph, and (ii) how we find a list-coloring in this conflict graph using the lists $L(\cdot)$. Throughout the proof, we condition on the high probability event in Lemma 4.1. We first show how to construct the conflict-graph. To do so, we rely on the by now standard primitive of ℓ_0-samplers for sampling elements in dynamic streams (see, e.g. [19, 27, 28]) captured in the following proposition.

[4]Here Δ is the maximum degree of the graph at the end of the stream and we assume no upper bound on degree of vertices throughout the stream, which can be as large as $\Omega(n)$ even when Δ is much smaller.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

PROPOSITION 4.1. (CF. [27, 36]) *There exists an streaming algorithm that given a subset $P \subseteq V \times V$ of pairs of vertices and an integer $k \geq 1$ at the beginning of a dynamic stream, outputs with high probability a set S of k edges from the edges in P that appear in the final graph (it outputs all edges if their number is smaller than k). The set S of edges can be either chosen uniformly at random with replacement or without replacement. The space of algorithm is $O(k \cdot \log^3 n)$.*

LEMMA 4.2. *$G_{\mathsf{conflict}}(V, E_{\mathsf{conflict}})$ can be constructed in $\widetilde{O}(n)$ space and polynomial time in dynamic streams with high probability.*

Proof. We construct the sets $\chi_1, \ldots, \chi_{\Delta+1}$ and store the sets in $O(n \log n)$ space. For any vertex $v \in V$, we define the set P_v of all edge slots between v and $\bigcup_{c \in L(v)} \chi_c$, i.e., all vertices that may have an edge to v in E_{conflict}, and run the algorithm in Proposition 4.1 with $P = P_v$ and parameter $k = O(\log^2(n))$.

As we conditioned on the event in Lemma 4.1, the degree of each vertex is at most k in G_{conflict}. Hence, by Proposition 4.1, with high probability, we find all neighbors of this vertex in G_{conflict}. Taking a union bound over all vertices in V, with high probability, we can construct the graph G_{conflict}. As all these steps can be implemented in polynomial time, we obtain the final result. ∎

Lemma 4.2 together with Lemma 4.1 are already enough to achieve a semi-streaming algorithm with exponential-time (i.e., prove Theorem 4.1 if we do not want a polynomial time algorithm): after constructing the conflict-graph G_{conflict}, we can simply use an exponential time algorithm to find a proper list-coloring of G_{conflict} which would be a $(\Delta+1)$ coloring of G by Lemma 4.1. We now show how to find a proper list-coloring of G_{conflict} in polynomial time.

A Polynomial Time Streaming Algorithm for $(\Delta+1)$ Coloring. Recall that the proof of Theorem 3.1 was in fact constructive modulo one part: we assumed the existence of the extended HSS-decomposition and found the $(\Delta+1)$ coloring using this assumption. However, without no explicit access to the extended HSS-decomposition, it is not clear how to find such a coloring. We remedy this by presenting a streaming algorithm that can compute this decomposition explicitly in a small space in one pass over the stream. We then use the argument in Theorem 3.1 to color the graph G in polynomial time.

LEMMA 4.3. *There exists a single-pass dynamic streaming algorithm that can find the extended HSS-decomposition of any given graph $G(V, E)$ with parameter $\varepsilon > 0$ in $\widetilde{O}(n/\varepsilon^2)$ space and polynomial time with high probability.*

The rest of this section is devoted to the proof of Lemma 4.3. Let ε be the parameter in Lemma 4.3 and define $\delta := \varepsilon/10$ throughout this section. We prove Lemma 4.3 in multiple steps. The first step is to determine for all vertices $u, v \in V$, whether these vertices can be δ-friend or not (as in Definition 1). Formally, we say that $u, v \in V$ are a *δ-potential friend* iff $|N(v) \cap N(u)| \geq (1-\delta)\Delta$. Note that the only difference between this definition and definition of δ-friends in Definition 1 is that here u and v may *not* be neighbor to each other. We design a simple data structure to determine (approximately) whether $u, v \in V$ are δ-potential friend or not in a single pass over the dynamic stream.

LEMMA 4.4. *There exists a single-pass dynamic streaming algorithm that constructs a data structure to answer the following queries: Given a pair of vertices $(u, v) \in V \times V$,*

- *outputs* Yes *if u and v are δ-potential friend;*
- *outputs* No *if u and v are not 2δ-potential friend;*
- *the answer can be arbitrary otherwise.*

The algorithm requires $\widetilde{O}(n/\delta^2)$ space and outputs the correct data structure with high probability.

Proof. The algorithm is simply as follows: Pick a set S of vertices at the beginning of the stream by choosing each vertex independently and with probability $p := \frac{10 \log n}{\delta^2 \Delta}$. For any chosen vertex in S, run the algorithm in Proposition 4.1 with P being the set of all edge slots incident to the vertex and $k = \Delta$. As the degree of every vertex is at most Δ, with high probability we recover all edges incident on S at the end of the stream. In the following, we condition on this event.

To answer the query (u, v) we do as follows: let $c_{u,v}$ be the number of common neighbors of u and v in S (as we have all the edges incident on S we can determine this quantity). Now output Yes iff $c_{u,v} \geq (1 - 1.5\delta) \cdot \Delta \cdot p$ and No otherwise. Let $N(u, v)$ denote the set of common neighbors of u and v. The proof of correctness is as follows:

- Suppose u and v are δ-potential friend and hence $|N(u, v)| \geq (1-\delta)\Delta$. In expectation, $\mathbb{E}[c_{u,v}] = |S \cap N(u, v)| \geq (1-\delta) \cdot \Delta \cdot p$ and hence by Chernoff bound, with probability at least $1 - 1/n^5$, $c_{u,v} \geq (1 - 1.5\delta) \cdot \Delta \cdot p$.
- Suppose u and v are not 2δ-potential friend and hence $|N(u, v)| < (1-2\delta)\Delta$. The above argument implies that $c_{u,v} < (1 - 1.5\delta) \cdot \Delta \cdot p$ with probability $1 - 1/n^5$.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

By taking a union bound over all n^2 (u,v)-pairs, the answer of the data structure is correct with high probability for all pairs.

Finally, another direct application of Chernoff bound ensures that the total number of vertices chosen in S is at most $O(n \log n/\Delta)$ with high probability. As each vertex requires $O(\Delta \cdot \text{polylog}(n))$ space to run the algorithm in Proposition 4.1, we obtain the bound on the space requirement of the algorithm as well. ∎

For the rest of the proof, we pick the data structure in Lemma 4.4 in our algorithm and assume that the high probability event in Lemma 4.4 happens. We now use this data structure to determine (approximately) if a vertex is δ-dense or not.

LEMMA 4.5. *There exists a single-pass dynamic streaming algorithm that uses $\widetilde{O}(n/\delta^2)$ space with high probability outputs a set D of vertices such that:*

- *any vertex in D is at least 2δ-dense;*
- *all $(\delta/2)$-dense vertices belong to D.*

Proof. We sample $k := \frac{100 \cdot n \log n}{\delta^2}$ edges E_s from the graph uniformly at random with replacement using Proposition 4.1. At the end of the stream, for any edge $(u,v) \in E_s$ we use the data structure in Lemma 4.4 on the pair (u,v) and discard this edge iff the answer to the query is No. Let E'_s be the set of non-discarded edges. We add any vertex u to D iff there are $\geq (1-\delta) \cdot \frac{k \cdot \Delta}{n \cdot \Delta}$ edges in E'_s incident on u.

Suppose first that u is a $(\delta/2)$-dense vertex and let F_u denote the set of $(\delta/2)$-friend edges incident on u. By Definition 2, $|F_u| \geq (1-(\delta/2))\Delta$. Moreover, by the guarantee of the data structure in Lemma 4.4, for any edge $(u,v) \in F_u$, the answer to the query (u,v) is Yes, as (u,v) is a $(\delta/2)$-friend edge (and hence is also $(\delta/2)$-potential friend by definition). Now let $c_u := |F_u \cap E_s|$ which is also a lower bound on the number of neighbors of u in E'_s. By the choice of E_s, we have that $\mathbb{E}[c_u] \geq (1-(\delta/2)) \cdot \frac{k \cdot \Delta}{n \cdot \Delta}$. An application of Chernoff bound now ensures that $c_u \geq (1-\delta) \cdot \frac{k \cdot \Delta}{n \cdot \Delta}$ with probability at least $1 - 1/n^5$. Hence, u would be added to D with probability at least $1 - 1/n^5$.

Now suppose u is not a 2δ-dense vertex. Let F_u denote the set of (2δ)-friend edges incident on u. By Definition 2, $|F_u| \leq (1-2\delta)\Delta$. By the guarantee of the data structure in Lemma 4.4, for any edge $(u,v) \notin F_u$, the answer to the query (u,v) in No as (u,v) in that case would not be a (2δ)-potential friend. Let $c_u := |F_u \cap E_s|$ which is an upper bound on the number of neighbors of u in E'_s. By the choice of E_s, we have $\mathbb{E}[c_u] \leq (1-2\delta)\frac{k \cdot \Delta}{n \cdot \Delta}$. As such, a Chernoff bound ensures that with probability at least $1 - 1/n^5$, u would not be added to D.

Taking a union bound over all n vertices finalizes the proof of correctness. The space needed by the algorithm is also $\widetilde{O}(n/\delta^2)$ by Proposition 4.1. ∎

We are now ready to construct the extended HSS-decomposition. Let D be the set of vertices found in Lemma 4.5 and for the rest of the proof, we assume that the high probability event in Lemma 4.5 happens. Define the following graph H with the set of vertices $V(H) := D$ and the set of edges:

$$E(H) := \{(u,v) \in E \mid \text{data structure in Lemma 4.4 returns Yes to the query } (u,v)\}.$$

Notice that we cannot construct the graph H explicitly and we only use it in the analysis. However, we can construt a graph H_s which is an edge-sampled subgraph of H. We will prove that H_s will have the same decomposition as H. Before showing how to construct H_s, we first recall a result about the connectivity of a subgraph obtained by sampling each edge of an original dense graph.

LEMMA 4.6. ([4]) *Let G_p be a subgraph of a graph G chosen by sampling each edge of G with probability p. For any constant $b > 0$, there exists a constant $c = c(b)$, such that if the minimum cut in G is of size at least $(c \log n/p)$, then G_p is connected with probability at least $1 - 1/n^b$.*

Let E_s be a collection of $\ell := \frac{10c \cdot n \log n}{\delta^2}$ (where c is the constant $c(b)$ in Lemma 4.6 with $b = 3$) edges chosen uniformly at random without replacement from the graph, which we pick in $\widetilde{O}(n/\delta^2)$ space using Proposition 4.1. Define the graph H_s with the set of vertices $V(H_s) := D$ and the set of edges:

$$E(H_s) := \{(u,v) \in E_s \mid \text{data structure in Lemma 4.4 returns Yes to the query } (u,v)\}.$$

We can construct the graph H_s explicitly at the end of the stream. We note that an important property of $E(H_s)$ is that any edge in $E(H)$ which is sampled in E_s also appears in $E(H_s)$, i.e., $E(H) \cap E_s \subseteq E(H_s)$; this is by the guarantee of Lemma 4.4.

We first prove that the connected components of H has similar properties to almost-cliques in the extended HSS-decomposition. The proofs of the following lemmas are postponed to the full version [9]. The following lemma is similar to Lemma 2.1.

LEMMA 4.7. *With high probability for any connected component C_i in H:*

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

1. $|C_i| \leq (1+6\delta)\Delta$;
2. *for any vertex v in C_i, we have at most $6\delta\Delta$ non-neighbors (in G) inside of C_i.*

The following lemma is similar to Lemma 2.2

LEMMA 4.8. *With high probability, any connected component C_i in H which has a $(\delta/4)$-dense vertex has at least $(1-\delta/4)\Delta$ vertices.*

By Lemma 4.8, we can see that any connected component of H which has a $(\delta/4)$-dense vertex has the same properties as the components of the extended HSS-decomposition in Lemma 2.2 (for some $\varepsilon = \Theta(\delta)$). Recall that however our streaming algorithm computes the graph H_s not H. In the following, we prove that this is not a problem as the connected components of H_s and H that contain a $(\delta/4)$-dense vertex are the same with high probability.

LEMMA 4.9. *With high probability, any connected component C_i in H which has a $(\delta/4)$-dense vertex is also connected in H_s.*

Proof. Fix any connected component C_i of H which contains a $(\delta/4)$-dense vertex. In the following, we condition on the events in Lemmas 4.7 and 4.8. By Lemma 4.8, $|C_i| \geq (1-\delta/4)\Delta \geq (1-\delta)\Delta$. Moreover, by Lemma 4.7, for any vertex $v \in C_i$, v has at most $6\delta\Delta$ non-neighbors inside of C_i. So any vertex $v \in C_i$ has at least $(1-7\delta)\Delta$ neighbors inside of C_i.

Fix any cut (V_1, V_2) of C_i with $|V_1| \leq |V_2|$. We first have $|V_1| \leq (1+6\delta)\Delta/2$ by the upper bound on size of C_i in Lemma 4.7. Moreover, any vertex $v \in V_1$ has at least $(1-7\delta)\Delta - |V_1|$ neighbors in V_2 by the lower bound on the number of neighbors of v in C_i, which means the size of the cut (V_1, V_2) is at least $|V_1|((1-7\delta)\Delta - |V_1|)$. By the choice of δ, we have $(1-7\delta)\Delta - |V_1| > 0$, so the cut size is at least $(1-7\delta)\Delta - 1 \geq 0.5\Delta$ with high probability.

Let c be the constant in Lemma 4.6 when b is 3. Define the subgraph H'_s as the subgraph of H by picking each edge in H with probability $\frac{2cn\log n}{\Delta}$ in a set E'_s (this is slightly different from H_s as in H_s we sample a fixed number of edges). By the above argument, since the minimum cut in H is at least 0.5Δ, by Lemma 4.6, the vertices in C_i are still connected in H'_s as well. On the other hand, recall that by the construction of H_s, E_s is a collection of $\frac{10cn\log n}{\Delta}$ randomly sampled edges without replacement. We call H_s (resp. H'_s) is good if (i) $|E_s|$ (resp. $|E'_s|$) is at most $\frac{10cn\log n}{\Delta}$, and (ii) any connected component C_i in H which has a $(\delta/4)$-dense vertex is also connected in H_s (resp. H'_s). By a straightforward coupling argument, the probability of H_s is good is at least the probability of H'_s is good. By above argument, any connected component C_i in H which has a $(\delta/4)$-dense vertex is also connected in H'_s. By Chernoff bound, the number of edges in E'_s is at most $\frac{10cn\log n}{\Delta}$ with high probability. So H'_s is good with high probability, which means H_s is also good with high probability. ∎

Let $C_1, C_2, \ldots, C_k$ be the connected components of H_s which have at least $(1-\delta)\Delta$ vertices, and $V_\star^{\text{sparse}}$ be $V \setminus C_1 \cup C_2 \cup \cdots \cup C_k$. The following lemma implies Lemma 4.3

LEMMA 4.10. *$V_\star^{\text{sparse}}, C_1, C_2, \ldots, C_k$ is with high probability a decomposition of V such that:*

1. *$V_\star^{\text{sparse}} \subseteq V_{\delta/4}^{\text{sparse}}$, i.e., any vertex in $V_\star^{\text{sparse}}$ is $(\delta/4)$-sparse.*
2. *For any $i \in [k]$, C_i has the following properties:*
   (a) *$(1-\delta)\Delta \leq |C_i| \leq (1+6\delta)\Delta$;*
   (b) *any $v \in C_i$ has at most $7\delta\Delta$ neighbors outside of C_i; and*
   (c) *any $v \in C_i$ has at most $6\delta\Delta$ non-neighbors inside of C_i.*

Proof. Since each connected component of H_s is a subset of a connected component of H, property 2 follows from Lemma 4.7 and the fact that $|C_i| \geq (1-\delta)\Delta$. On the other hand, by Lemma 4.8 and Lemma 4.9, any connected component of H_s which contains a $(\delta/4)$-dense vertex has at least $(1-\delta)\Delta$ vertices with high probability. This in turn implies that no $(\delta/4)$-dense vertex is in $V_\star^{\text{sparse}}$ with high probability. ∎

With the decomposition, we are now ready to conclude the proof of Theorem 4.1.

Proof. [Proof of Theorem 4.1] By spending $\widetilde{O}(n/\varepsilon^2)$ space throughout the stream, we obtain the decomposition given by Lemma 4.10. In the following, we condition on this event and use this decomposition to color G using only the sampled colors for vertices. The proof consists of three phases, which correspond to the three phases in the proof of Theorem 3.1.

Coloring Sparse Vertices. We use the process given by the proof of Lemma 3.1 to color the sparse vertices in $V_\star^{\text{sparse}}$ given by Lemma 4.10. We run GreedyColor algorithm with $O(\log n)$ rounds. In each round, each vertex v in $V_\star^{\text{sparse}}$ which have not been colored picks a color in $L_1(v)$. Then we check if the color is different from all the vertices in $N(v)$. If so, we color v by the chosen color. So we only need to iterate over the edges in G_{conflict}, which takes $O(n\log^2 n)$ time. Hence, this phase of the algorithm takes $O(n\log^3 n)$ time.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Initial Coloring of Almost Cliques. In this phase, for each almost-clique C_i given by Lemma 4.10, we find a colorful matching (as in Definition 3) given by the the second batch of sampled colors L_2. We use the process given by the proof of Lemma 3.2. For each color, if we can find a pair of vertices u, v such that (u, v) is not in G_{conflict}, u and v are not in the colorful matching yet, and $L(u)$ and $L(v)$ both contain the same color, then we add (u, v) with this color to the colorful matching. Hence, this phase also takes $\widetilde{O}(n)$ time.

Final Coloring of Almost-Cliques. In this phase, we color the remaining vertices inside almost-cliques. To color these vertices, we color the almost-cliques one by one. When coloring almost-clique C_i, we construct the palette-graph (as in Definition 4) between the vertices and the colors, and find a maximum matching of this graph. By Lemma 3.4 there is a matching that pairs each vertex to an available color. To construct the *palette-graph*, we need to connect each vertex v with all colors in $L_3(v)$. Then we iterate over the edges of G_{conflict}, delete the edges between a vertex and an unavailable color. The construction of the palette-graph for one almost-clique takes $O(\Delta \log^2 n)$ time. Finding the matching also require $O(\Delta^{3/2})$ time by using the standard Hopcraft-Karp algorithm [25] for bipartite matching. There are at most $O(n/\Delta)$ near-cliques in G, so this phase takes at most $\widetilde{O}(n\sqrt{\Delta})$ time with high probability. ∎

4.2 A Sublinear Time Algorithm for $(\Delta+1)$ Coloring We now show another application of our palette-sparsification theorem to design sublinear algorithms. Consider the following standard query model for sublinear time algorithms on general graphs (see, e.g., Chapter 10 of Goldreich's book [21]): The vertex set of the graph is $V := [n]$ and the algorithm can make the following queries: (i) Degree queries: given $v \in V$, outputs degree $d(v)$ of v, (ii) Neighbor queries: given $v \in V$ and $i \leq d(v)$, outputs the i-th neighbor of v (the ordering of neighbors are arbitrary), and (iii) Pair queries: given $u, v \in V$, outputs whether the edge (u, v) is in E or not. We give a sublinear time algorithm (in size of the graph) for finding a $(\Delta+1)$ coloring in this query model.

THEOREM 4.2. *There exists an algorithm that given a query access to a graph $G(V, E)$ with maximum degree Δ, can find a $(\Delta+1)$ coloring of G with high probability in $\widetilde{O}(n\sqrt{n})$ time and queries.*

We prove Theorem 4.2 by combining two separate algorithms and picking the best of the two depending on the value of Δ. One is the straightforward (deterministic) greedy algorithm that takes $O(n\Delta)$ time to find a $(\Delta+1)$ coloring. This algorithm only uses neighbor queries. We use this algorithm when $\Delta \leq \sqrt{n}$. The other one is an implementation of ColoringAlgorithm in the query model which takes $\widetilde{O}(n^2/\Delta)$ time. This algorithm only uses pair queries. By using this algorithm when $\Delta \geq \sqrt{n}$, we achieve an $\widetilde{O}(n\sqrt{n})$ time algorithm for any graph (with potentially $\Omega(n^2)$ edges), proving Theorem 4.2.

To implement ColoringAlgorithm, we need to specify (i) how to construct the conflict-graph, and (ii) how to find a list-coloring in this conflict graph using the lists $L(\cdot)$. Throughout the proof, we condition on the high probability event in Lemma 4.1. The first part of the argument is quite easy as is shown below.

CLAIM 4.11. *$G_{\mathsf{conflict}}(V, E_{\mathsf{conflict}})$ can be constructed in $O(n^2 \cdot \log^2 n/\Delta)$ queries and time.*

Proof. As the vertices are known, we only need to construct the edges E_{conflict}. In order to do this, we simply query all pairs between vertices inside each set χ_c for $c \in [\Delta+1]$. By Lemma 4.1, size of each χ_c is $O(n \log n/\Delta)$ and hence we need $O(n^2 \log^2 n/\Delta^2 \cdot (\Delta+1)) = O(n^2 \log^2 n/\Delta)$ queries. ∎

Claim 4.11 is already sufficient to obtain an $\widetilde{O}(n^2/\Delta)$ *query* (but not time) algorithm: by Lemma 4.1, ColoringAlgorithm outputs the correct answer by finding a list-coloring of G_{conflict} and accessing G_{conflict} does not require further queries to G. However, finding such a list-coloring problem in general is NP-hard and hence to find this coloring in sublinear time, we need to design an algorithm which further queries the graph G to obtain additional information for performing the coloring. The idea is as in the previous section by finding an (approximate) extended-HSS decomposition of G using a small number of queries and then constructing the list-coloring of G_{conflict} using this additional information.

List-Coloring G_{conflict} in Sublinear Time in the Query Model. Just as in the case of the streaming model, we first give an algorithm which finds an approximate extended-HSS decomposition of G in $\tilde{O}(n^2/\Delta)$ time. We then use an arguments similar to Theorem 3.1 to color graph G also in $\tilde{O}(n^2/\Delta)$ time. The parameter δ below is as defined in Section 4.1.

To find an approximate extended-HSS decomposition, we use similar ideas as in Section 4.1. We first give a data structure to determine whether a pair of vertices are potential friends (see Section 4.1 for the definition). We then use this data structure to find dense vertices. The arguments are similar to Lemma

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

4.4 and Lemma 4.5, but we now implement them in sublinear time using only pair queries. Once we have these two lemmas, we define the graph H on dense vertices as in Section 4.1 and find a random subgraph H_s of H by random sampling edges, and prove that the subgraph has the same connected components as H, which allows us to recover the decomposition.

The following two lemmas are analogies to Lemma 4.4 and Lemma 4.5, but we achieve them in sublinear time in query model.

LEMMA 4.12. *There exists a data structure that can be created in $\tilde{O}(n^2/\Delta)$ time using pair queries for answering the following queries: Given a pair of vertices $(u, v) \in V \times V$,*

- *outputs* Yes *if u and v are δ-potential friend;*
- *outputs* No *if u and v are not 2δ-potential friend; and*
- *the answer can be arbitrary otherwise.*

For each query pair u, v, the data structure can be used to recover the correct answer with high probability in $O(\log n)$ time.

Proof. We use the same notation as in Lemma 4.4. To construct the data structure, pick a set of S of vertices by choosing each vertex with probability $p = \frac{10 \log n}{\delta^2 \Delta}$. For each vertex $v \in S$, use the pair queries to find all its neighbors in G, and construct a linked list for each vertex $u \in V$ to store all the neighbors of u in S. Now to answer the query of a pair of vertices u and v, use the list of neighbors in the data structure to calculate $c_{u,v} := |N(u) \cap N(v) \cap S|$, i.e., the number of common neighbors of u and v in S. Output Yes if and only if $c_{u,v} \geq (1 - 1.5\delta)\Delta \cdot p$.

Using the proof of Lemma 4.4, the answer of each query is correct with high probability. So we only need to show the running time of the algorithm. By Chernoff bound, the number of vertices chosen in S is at most $O(n \log n/\Delta)$ with high probability. We query all pairs of vertices which contains the vertices in S. It takes $O(n^2 \log n/\Delta)$ queries. We can construct the linked lists during the queries. So we only need $O(n^2 \log n/\Delta)$ time to construct the data structure with high probability. For each vertex, there are at most Δ neighbors in G and each neighbor is chosen in S with probability p. Hence, by Chernoff bound, there are at most $O(p\Delta) = O(\log n)$ neighbors in S with high probability. So for each query, we need at most $O(\log n)$ time to calculate $c_{u,v}$ and answer the query with high probability. ∎

From now on, we condition on the event in Lemma 4.12 for all pair of vertices, which still happens with high probability by a union bound. We can now use the above data structure to find δ-dense vertices approximately.

LEMMA 4.13. *There exists an algorithm that uses $\widetilde{O}(n^2/\Delta)$ time and pair queries and with high probability outputs a set D of vertices such that:*

- *any vertex in D is at least 2δ-dense;*
- *all $(\delta/2)$-dense vertices belong to D.*

Proof. The proof is again similar to the proof of Lemma 4.5. We add each edge in the graph to a set E_s with probability $p = \frac{10 \log n}{\delta^2 \Delta}$ independently. To construct the set E_s, we simply need to sample every edge-slot in the graph with probability p and then query the sampled edge-slot. With high probability, this can be done in $O(n^2 \cdot p) = O(n^2 \log n/\Delta)$ pair queries (to sample this set, we can sample random variables by geometric distribution with parameter p to determine the next pair to query).

For each edge $(u, v) \in E_s$, we use the data structure in Lemma 4.12 to determine if they are potential friends. We discard (u, v) from E_s if the answer of the query is No. We can do it in $O(n^2 \log^2 n/\Delta)$ time with high probability since each query takes $O(\log n)$ time (as we conditioned on the event in Lemma 4.12). Add a vertex v to S if there are at least $(1 - 1.5\delta)\Delta p$ edges in E_s which is incident on v. By the same argument as in the proof of Lemma 4.5, the output S satisfies the requirements in the lemma with high probability. ∎

For the rest of the proof, we further condition on the event in Lemma 4.13. Recall the definition of the graph H from Section 4.1. We can now construct a random subgraph H_s of H. Let E_s be the edge set by choosing each edge in the graph with probability $p = \frac{4c \log n}{\Delta}$, where c is the constant $c(b)$ in Lemma 4.9 with $b = 3$. Let H_s be the graph that only contains vertices in D (the output set of Lemma 4.13), with the edge set:

$$E(H_s) = \{(u, v) \in E_s | \text{data structure in Lemma 4.12 returns Yes to query } (u, v)\}.$$

To construct the graph H_s, we need $O(n^2 p) = O(n^2 \log n/\Delta)$ time with high probability to construct E_s, and $O(n^2 \log^2 n/\Delta)$ time to query each edge in E_s using the data structure in Lemma 4.12.

Let $C_1, C_2, \ldots, C_k$ be the connected components of H_s which have at least $(1 - \delta)\Delta$ vertices, and $V_\star^{\text{sparse}}$ be $V \setminus C_1 \cup C_2 \cup \cdots \cup C_k$. The following lemma directly follows from the proof of Lemma 4.10 with Lemma 4.12 and Lemma 4.13.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

LEMMA 4.14. *With high probability, we can compute a decomposition of* $V = V_\star^{\text{sparse}} \cup C_1 \cup C_2 \cup \cdots \cup C_k$ *in* $\widetilde{O}(n^2/\Delta)$ *time and pair queries such that:*

1. $V_\star^{\text{sparse}} \subseteq V_{\delta/4}^{\text{sparse}}$*, i.e., any vertex in* $V_\star^{\text{sparse}}$ *is* $(\delta/4)$*-sparse.*
2. *For any* $i \in [k]$*,* C_i *has the following properties:*
   (a) $(1-\delta)\Delta \leq |C_i| \leq (1+6\delta)\Delta$*;*
   (b) *any* $v \in C_i$ *has at most* $7\delta\Delta$ *neighbors outside of* C_i*; and*
   (c) *any* $v \in C_i$ *has at most* $6\delta\Delta$ *non-neighbors inside of* C_i*.*

Proof of Theorem 4.2 can now be concluded exactly as in the proof of Theorem 4.1, as we show in that proof that starting from the decomposition, one can construct the coloring in $\widetilde{O}(n\sqrt{n})$ time.

4.3 An MPC Algorithm for $(\Delta+1)$ Coloring

This section contains yet another application of our palette-sparsification theorem to design sublinear algorithms, namely a massively parallel (MPC) algorithm for $(\Delta+1)$ coloring. In the MPC model of [12] (see also [6,12,22,29]), the input is partitioned across multiple machines which are inter-connected via a communication network. The computation proceeds in synchronous rounds. During a round each machine runs a local algorithm on the data assigned to the machine. No communication between machines is allowed during a round. Between rounds, machines are allowed to communicate so long as each machine sends or receives a communication no more than its memory. Any data output from a machine must be computed locally from the data residing on the machine and initially the input data is distributed across machines adversarially. The goal is to minimize the total number of rounds subject to a small (sublinear) memory per machine.

We show that ColoringAlgorithm can be easily implemented in this model also and prove the following theorem.

THEOREM 4.3. *There exists a randomized MPC algorithm that given a graph* $G(V,E)$ *with maximum degree* Δ *can find a* $(\Delta+1)$ *coloring of* G *with high probability in* $O(1)$ *MPC rounds on machines of memory* $\widetilde{O}(n)$*. Furthermore, if the machines have access to* public randomness*, the algorithm only requires* one *MPC round.*

In the following, we assume that the machines have access to public randomness and show how to solve the problem in only one MPC round (this setting is similar to the result of [3] for the connectivity problem and maybe of independent interest). We then show that by spending $O(1)$ additional rounds, we can remove the assumption of public randomness.

The proof of this theorem is very similar to that of Theorem 4.1 and uses the close connection between dynamic streaming algorithms (in particular linear sketching algorithms) and MPC algorithms (see, e.g. [2, 3]). As before, if we do not insist on achieving a polynomial time algorithm, proving Theorem 4.3 from Lemma 4.1 is straightforward: we sample the color classes $\chi_1, \ldots, \chi_{\Delta+1}$ using public randomness, and every machine sends its edges in G_{conflict} to a central designated machine, called the coordinator. As the total number of edges in G_{conflict} is $\widetilde{O}(n)$ by Lemma 4.1, we only need $\widetilde{O}(n)$ memory on the coordinator. The coordinator machine can then locally find a list-coloring of the graph G_{conflict} and find the $(\Delta+1)$ coloring by Lemma 4.1.

We now briefly show how to make this algorithm polynomial time. The proof of this part is identical to that of Theorem 4.1: we can construct a decomposition of the graph G in Lemma 4.10 exactly as in Section 4.1 on the coordinator machine and then run the polynomial time algorithm in the proof of Theorem 4.1 on this decomposition to solve the problem. In order to see this, note that the decomposition in Lemma 4.10 relied on vertex sampling (to implement the data structure in Lemma 4.4) and edge sampling (to implement the data structure in Lemma 4.5 as well as defining and finding the connected components of the graph H_s), both of which can be done trivially in the MPC model using public randomness.

Finally, we show how to remove the public randomness. We first dedicate one machine M_v to each vertex v of the graph and spend the first round to send all the edges incident on v to the machine M_v. This can be done on machines of memory $O(\Delta)$. In the next round, each machine v samples the set of colors $L(v)$ for v and sends this information to all the machines M_u where (u,v) is an edge in the graph. This can again be done with $O(\Delta \cdot \text{polylog}(n))$ size messages and hence on machines of memory $\widetilde{O}(n)$. The machines can now send all edges in E_{conflict} to a central coordinator and the coordinator can construct the graph G_{conflict}. Finally, to find the decomposition in Lemma 4.10 also, each machine M_v can do the sampling of vertex v locally and send this vertex and all its incident edges (as needed in the proof of Lemma 4.4) to the central coordinator; the edge sampling can also be done trivially as each machine holding an edge can decide to sample it with probability p and send it to the coordinator independently (as needed in the proof of Lemma 4.5 and also in finding connected components of H_s). All in all, we only need $O(1)$ MPC rounds to run the algorithm.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

5 Lower Bounds for Sublinear Graph Coloring and Related Problems

In this section, we prove new lower bounds for maximal independent set (MIS) and maximal matching problems in the considered models to contrast the complexity of the $(\Delta+1)$ coloring problem with these two closely related problems

THEOREM 5.1. (Streaming MIS) *Any single-pass streaming algorithm that is able to output any arbitrary MIS of the input graph in insertion-only streams with sufficiently large constant probability requires $\Omega(n^2)$ space.*

Theorem 5.1 implies that unlike the $(\Delta+1)$ coloring problem, one cannot obtain any non-trivial single-pass streaming algorithm for the maximal independent set even in insertion-only streams.

THEOREM 5.2. (Sublinear-Time Maximal Matching) *Any algorithm (possibly randomized) that can output a maximal matching of an input graph with sufficiently large constant probability requires $\Omega(n^2)$ queries to the graph.*

Theorem 5.2 implies that unlike the $(\Delta + 1)$ coloring problem, one cannot hope to obtain any non-trivial sublinear-time algorithm for the maximal matching problem.

We also prove a lower bound for $(\Delta+1)$ coloring problem in the query model that shows that our algorithm in Theorem 4.2 achieves optimal query/time complexity (up to logarithmic factors). This lower bound holds even for algorithms that are required to output an $O(\Delta)$ coloring not only $(\Delta+1)$.

THEOREM 5.3. (Sublinear Time Vertex Coloring) *For any fixed constant $c > 1$, any algorithm (possibly randomized) that can output a $(c \cdot \Delta)$ coloring of an input graph with sufficiently large constant probability requires $\Omega(n\sqrt{n})$ queries to the graph.*

We prove Theorems 5.1 and 5.2 in Sections 5.1 and 5.2. The proof of Theorem 5.3 is more technical and is postponed to the full version of the paper [9].

5.1 A Lower Bound for Single-Pass Streaming MIS

We prove Theorem 5.1 in this section. Throughout this section, we define MIS as the two-player communication problem in which the edge-set E of a graph $G(V, E)$ is partitioned between Alice and Bob, and their goal is to find *any* MIS of the graph G. To prove Theorem 5.1, it suffices to lower bound the *one-way* communication complexity of MIS. We do this using a reduction from the Index problem. In Index, Alice is given a bit-string $x \in \{0,1\}^N$, Bob is given an index $k^* \in [N]$, and the goal is for Alice to send a message to Bob so that Bob outputs x_{k^*}. It is well-known that one-way communication complexity of Index is $\Omega(N)$ bits [1, 31].

To continue, we need a simple definition. Let $n := \sqrt{N}$ and throughout the proof, fix any arbitrary bijection $\sigma : [N] \mapsto [n] \times [n]$. For any bit-string $x \in \{0,1\}^N$, and sets $A := \{a_1, \ldots, a_n\}$ and $B := \{b_1, \ldots, b_n\}$, we define the *$x$-graph* G on vertices $A \cup B$ as a bipartite graph consisting of all edges (a_i, b_j) where $x_k = 1$ and $\sigma(k) = (i, j)$.

Reduction. Given an instance (x, k^*) of Index:

1. Alice and Bob construct the following graph $G(V, E)$ with *no* communication:
   - The vertex set V of the graph G is the union of the following four sets of vertices:
     $$A_1 := \left\{a_1^1, \ldots, a_n^1\right\}, B_1 := \left\{b_1^1, \ldots, b_n^1\right\},$$
     $$A_2 := \left\{a_1^2, \ldots, a_n^2\right\}, B_2 := \left\{b_1^2, \ldots, b_n^2\right\}.$$
   - Alice forms two x-graphs G_1 and G_2 on $A_1 \cup B_1$ and $A_2 \cup B_2$, respectively, using her bit-string x and adds these edges to E.
   - Let $(i^*, j^*) = \sigma(k^*)$ where k^* is the input index of Bob. Bob adds the following three sets of edges to E:
     $$\text{(for all } i \neq i^* \in [n] \text{ and } j \neq i^* \in [n]) \quad \left\{(a_i^1, a_j^1), (a_i^2, a_j^2), (a_i^1, a_j^2)\right\},$$
     $$\text{(for all } i \neq j^* \in [n] \text{ and } j \neq j^* \in [n]) \quad \left\{(b_i^1, b_j^1), (b_i^2, b_j^2), (b_i^1, b_j^2)\right\},$$
     $$\text{(for all } i \neq j^* \in [n] \text{ and } j \neq j^* \in [n]) \quad \left\{(a_i^1, b_j^2), (a_i^2, b_j^1)\right\}.$$
2. Alice and Bob then compute any MIS $\mathcal{M}$ of the graph G using the best protocol for MIS. Bob outputs $x_{k^*} = 0$ if either both $a_{i^*}^1, b_{j^*}^1 \in \mathcal{M}$ or both $a_{i^*}^2, b_{j^*}^2 \in \mathcal{M}$.

This finalizes the description of the reduction. It is immediate to verify that the communication cost of this protocol is at most as large as the communication complexity of MIS. In the following two claims, we prove the correctness of this protocol.

CLAIM 5.1. *Let G be the graph constructed by Alice and Bob, and $\mathcal{M}$ be any MIS of G. Then, if $x_{k^*} = 1$, $\mathcal{M}$ cannot contain both $a_{i^*}^1, b_{j^*}^1$, neither both $a_{i^*}^2, b_{j^*}^2$.*

Proof. If $x_{k^*} = 1$, then $(a_{i^*}^1, b_{j^*}^1) \in G_1$ and $(a_{i^*}^2, b_{j^*}^2) \in G_2$ by construction of the x-graph. ∎

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

CLAIM 5.2. *Let G be the graph constructed by Alice and Bob. Assuming $x_{k^*} = 0$, any MIS $\mathcal{M}$ of G contains either both $a^1_{i^*}, b^1_{j^*}$ or both $a^2_{i^*}, b^2_{j^*}$.*

Proof. Suppose towards a contradiction that $\mathcal{M}$ does not satisfy the assertion of the claim statement. It is easy to see that no collection of two vertices from $a^1_{i^*}, b^1_{j^*}, a^2_{i^*}, b^2_{j^*}$ can be a dominating set in G and an MIS (any MIS of a graph is a dominating set). This implies that $\mathcal{M}$ necessarily contains a vertex $v \in G \setminus \{a^1_{i^*}, b^1_{j^*}, a^2_{i^*}, b^2_{j^*}\}$. For simplicity, let us assume that $v \in A_1$; the other cases hold by symmetry. We know that Bob has connected v to all vertices in $A_1 \cup A_2 \setminus \{a^1_{i^*}, a^2_{i^*}\}$ as well as vertices in $B_2 \setminus b^2_{j^*}$. As such, none of these vertices can be part of $\mathcal{M}$.

Now consider vertices $a^2_{i^*}, b^2_{j^*}$. All "potential" neighbors of $a^2_{i^*}$ in G are vertices in B_2 and hence except for $b^2_{j^*}$ none of them can be in $\mathcal{M}$. Similarly, all potential neighbors of $b^2_{j^*}$ in G are vertices in A_2 and again except for $a^2_{i^*}$ none of them can be in $\mathcal{M}$. Maximality of MIS plus the fact that there are no edges between $a^2_{i^*}$ and $b^2_{j^*}$ (as $x_{k^*} = 0$), implies that both $a^2_{i^*}$ and $b^2_{j^*}$ are in $\mathcal{M}$, a contradiction. ∎

We are now ready to prove Theorem 5.1.

Proof. [Proof of Theorem 5.1] Let π be any 1/3-error one-way protocol for MIS. By Claims 5.1 and 5.2, we obtain a protocol for Index which errs with probability at most 1/3 and has communication cost at most equal to cost of π. By the $\Omega(N)$ lower bound on the one-way communication complexity of Index, we obtain that communication cost of π must be $\Omega(N) = \Omega(n^2)$. As the total number of vertices in the graph is $O(n)$, we obtain an $\Omega(n^2)$ lower bound on the communication complexity of MIS. Theorem 5.1 now follows from this as one-way communication complexity lower bounds the space complexity of single pass streaming algorithms. ∎

5.2 A Lower Bound for Sublinear Time Maximal Matching We prove Theorem 5.2 in this section. Recall the definition of the query model from Section 4.2. We will show that there exists a family of n-vertex graphs such that any randomized algorithm for finding a maximal matching requires $\Omega(n^2)$ queries in expectation and use this to prove Theorem 5.2. We will show that this lower bound holds even for bipartite graphs.

We will denote our input graph by $G(V, E)$ where V is bipartitioned into sets L and R. We will assume that the algorithm is given degrees of all the vertices for free. As such, we simply consider the following two types of queries:

- $\mathbf{Q}_1(u, i)$ (neighbor query) : given any vertex $u \in V$ and an integer $i \in [1..\deg(u)]$, this query returns the i-th neighbor of u in the adjacency list of u (the ordering of the list is arbitrary).
- $\mathbf{Q}_2(u, v)$ (pair query): given a pair of vertices $u, v \in V$, this query returns 0/1 to indicate absence or presence of the edge (u, v).

By Yao's minimax principle [42], it suffices to create a distribution over n-vertex graphs such that any deterministic algorithm requires $\Omega(n^2)$ queries to find a maximal matching on this distribution. A graph from our distribution is generated as follows. The set L is partitioned into sets L_1 and L_2, and the set R is partitioned into sets R_1 and R_2 with $|L_1| = |R_1| = n/6$, and $|L_2| = |R_2| = 5n/6$. Each vertex u in L is connected to every vertex in R_1, and each vertex v in R is connected to each vertex in L_1. In addition, we add to G a random perfect matching M between L_2 and R_2. Finally, the adjacency list of each vertex is created by taking a random permutation of its neighbors. This completes the description of the process that generates our input graph $G(V, E)$.

Note that the final graph G contains a perfect matching of size n. Thus any maximal matching in G includes at least $n/3$ edges from M, and any maximal matching algorithm needs to discover at least $n/3$ edges in M. The heart of our lower bound argument is to show that this requires $\Omega(n^2)$ queries in expectation.

We assume that the algorithm is explicitly given the partition of vertices into sets L_1, L_2, R_1, and R_2. This means that the algorithm already knows adjacency list of each vertex $u \in L_1$ and each vertex $v \in R_1$, as well as the answer to each query $Q_2(u, v)$ whenever $u \in L$ and $v \in R_1$ or whenever $u \in L_1$ and $v \in R$. Thus the only potentially useful queries for any algorithm are of the form $Q_1(u, i)$ for some $u \in L_2 \cup R_2$ and $i \in [n/6+1]$, and $Q_2(u, v)$ for $u \in L_2$ and $v \in R_2$.

We say that an edge $(u, v) \in M$ has been *discovered* if the set of queries performed thus far uniquely identify the edge (u, v) to be in M. For any discovered edge $(u, v) \in M$, we will say the vertices u and v have been *discovered*; any vertex in $L_2 \cup R_2$ that has not been discovered is called an *undiscovered* vertex. We are now ready to prove the main result below.

LEMMA 5.3. *Given a graph G generated by the distribution described above, any deterministic algorithm requires $\Omega(n^2)$ queries in expectation to discover $n/3$ edges in M.*

Proof. After t queries have been made by the algorithm, let $L_U(t) \subseteq L_2$ and $R_U(t) \subseteq R_2$ denote the set

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

of undiscovered vertices in L_2 and R_2 respectively. Let $E(t) \subseteq L_U(t) \times R_U(t)$ denote the set of edge slots that have not yet been queried/discovered. Note that by our process for generating G, the undiscovered edges in M correspond to a random perfect matching between $L_U(t)$ and $R_U(t)$ that is entirely supported on $E(t)$.

We will analyze the performance of any algorithm by partitioning the queries into phases. The first query by the algorithm starts the first phase, and a phase ends as soon as an edge in M has been discovered. Let Z_i be a random variable that denotes the number of queries performed in phase i of the algorithm. Thus we wish to analyze the $E[\sum_{i=1}^{n/3} Z_i]$. The following lemma is crucial to our analysis (the proof appears in full version [9]).

LEMMA 5.4. *Let $G'(L' \cup R', E')$ be an arbitrary bipartite graph such that $|L'| = |R'| = N$, and each vertex in G' has degree at least $2N/3$. Then for any edge $e = (u, v) \in E'$, the probability that e is contained in a uniformly at random chosen perfect matching in G' is at most $3/N$.*

For any vertex $w \in (L_U(t) \cup R_U(t))$, we say that Q_1-*uncertainty* of w is d if there are at least d entries in the adjacency list of w that have not yet been probed. Similarly, for any vertex $w \in (L_U(t) \cup R_U(t))$, we say that Q_2-*uncertainty* of w is d if there are at least d edges in $E(t)$ that are incident on w.

At time t, we say a vertex $w \in (L_U(t) \cup R_U(t))$ is *bad* if either Q_1-uncertainty of w is less than $n/12$ or Q_2-uncertainty of w is less than $3n/4$. Note that if at some time t none of the vertices in $(L_U(t) \cup R_U(t))$ are bad then in the next $n/24$ time steps, any single Q_1 query succeeds in discovering an edge in M with probability at most $24/n$. Also, for the next $n/24$ time steps, the degree of any vertex in $L_U(t) \cup R_U(t)$ in $E(t)$ remains above $2n/3$. Thus by Lemma 5.4, the probability that any single Q_2 query made during the first $n/12$ queries in the phase succeeds (in discovery of a new edge in M) is at most $3/n$.

We say a phase is *good* if at the start of the phase, there are no bad vertices, and the phase is *bad* otherwise.

PROPOSITION 5.1. *The expected length of a good phase is at least $n/96$.*

Proof. If at the start of the phase i, no vertex is bad, then for the next $n/24$ time steps, the probability of success for any Q_1/Q_2 query is at most $24/n$. Thus the expected number of successes (discovery of a new edge in M) in the first $n/48$ time steps in a phase is at most $1/2$. By Markov's inequality, it then follows that with probability at least $1/2$, there are no successes among the first $n/48$ queries in a phase. Thus the expected length of the phase is $\geq n/96$. ∎

Note that if all phases were good, then it immediately follows that the expected number of queries to discover $n/3$ edges is $\Omega(n^2)$. To complete the proof, it remains to show that most phases are good. For ease of analysis, we will give the algorithm additional information for free and show that it still needs $\Omega(n^2)$ queries in expectation even to discover the first $n/24$ edges in M.

Whenever algorithm starts a bad phase, we immediately reveal to the algorithm an undiscovered edge in M that is incident on an arbitrarily chosen bad vertex. Thus each bad phase is guaranteed to consume a bad vertex (i.e., make the bad vertex discovered and hence remove it from further consideration). On the other hand, to create a bad vertex w, one of the following two events needs to occur: (a) either we have done at least $(n/12)$ Q_1 queries incident on w, or (b) the number of discovered edges in M plus the number of Q_2 queries is at least $5n/6 - 3n/4 = n/12$.

Since we are restricting ourselves to analyzing the discovery of first $n/24$ edges in M, any vertex w that becomes bad due to (b) above requires at least $n/24$ Q_2 queries incident on it. Thus to create K bad vertices in the first $n/24$ phases, we need to perform at least $(K \cdot (n/24))/2$ queries; here the division by 2 accounts for the fact that each Q_2 query reduces uncertainity for two vertices. It now follows that if the algorithm encounters at least $n/48$ bad phases among the first $n/24$ phases, then $K \geq n/48$ and hence it must have already performed $n^2/(48)^2$ queries. Otherwise, at least $n/48$ phases among the first $n/24$ phases are good, implying that the expected number of queries is at least $(n/48) \cdot (n/96)$. This completes the proof of Lemma 5.3. ∎

Proof. [Proof of Theorem 5.2] Suppose towards a contradiction that $\mathcal{A}$ is a query algorithm that uses $o(n^2)$ queries and find a maximal matching of the graph with probability $1 - o(1)$. Consider the distribution of instances introduced in this section and notice that any maximal matching necessarily needs to find $n/3$ edges from M (as any maximal matching is a 2-approximate matching). We run $\mathcal{A}$ on the distribution of this section and if it did not discover $n/3$ edges from M, we simply query all edges of the graph. As such, the expected query complexity of this algorithm before finding $n/3$ edges from M is $o(n^2) \cdot (1 - o(1))$ (if it finds the maximal matching) plus $n^2 \cdot o(1)$ (if it did not find the maximal matching and we queried all graph) which is $o(n^2)$. By fixing

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

the randomness of this algorithm against this distribution, i.e., using (the easy direction of) Yao's minimax principle, we obtain a deterministic algorithm that makes $o(n^2)$ queries in expectation and finds $n/3$ edges from M. This contradicts Lemma 5.3, which finalizes the proof. ∎

Acknowledgements. We thank the anonymous reviewers for many helpful comments.

References

[1] F. M. Ablayev. Lower bounds for one-way probabilistic communication complexity. In *Automata, Languages and Programming, 20nd International Colloquium, ICALP93, Lund, Sweden, July 5-9, 1993, Proceedings*, pages 241–252, 1993.

[2] K. J. Ahn and S. Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 202–211, 2015.

[3] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467. SIAM, 2012.

[4] N. Alon. A note on network reliability. In *Discrete Probability and Algorithms*, pages 11–14. Springer, 1995.

[5] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139, 2012.

[6] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014.

[7] S. Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, 2017.

[8] S. Assadi, M. Bateni, A. Bernstein, V. S. Mirrokni, and C. Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076. To appear in SODA 2019, 2017.

[9] S. Assadi, Y. Chen, and S. Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring. *CoRR*, abs/1807.08886, 2018.

[10] S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364, 2016.

[11] L. Barenboim and M. Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 129–138, 2011.

[12] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 273–284, 2013.

[13] S. Behnezhad, M. Derakhshan, and M. Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *CoRR*, abs/1802.10297, 2018.

[14] Y. Chang, W. Li, and S. Pettie. An optimal distributed $(\Delta+1)$-coloring algorithm? In *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, 2018.

[15] A. Czumaj, J. Lacki, A. Madry, S. Mitrovic, K. Onak, and P. Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 471–484, 2018.

[16] M. Elkin, S. Pettie, and H. Su. $(2\Delta - 1)$-edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 355–370, 2015.

[17] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

[18] M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191, 2017.

[19] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

[20] M. Ghaffari, T. Gouleakis, S. Mitrovic, and R. Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. *CoRR*, abs/1802.08237, 2018.

[21] O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

[22] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 374–383, 2011.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

[23] D. G. Harris, J. Schneider, and H.-H. Su. Distributed (Δ+ 1)-coloring in sublogarithmic rounds. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 465–478. ACM, 2016.

[24] N. J. A. Harvey, C. Liaw, and P. Liu. Greedy and local ratio algorithms in the mapreduce model. *CoRR*, abs/1806.06421. To appear in SPAA 2018, 2018.

[25] J. E. Hopcroft and R. M. Karp. A nˆ5/2 algorithm for maximum matchings in bipartite graphs. In *12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971*, pages 122–125, 1971.

[26] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108, 2014.

[27] H. Jowhari, M. Sağlam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58. ACM, 2011.

[28] M. Kapralov, J. Nelson, J. Pachocki, Z. Wang, D. P. Woodruff, and M. Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486, 2017.

[29] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010.

[30] C. Konrad. MIS in the congested clique model in o(log log Δ) rounds. *CoRR*, abs/1802.07647, 2018.

[31] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 596–605, 1995.

[32] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 1–10, 2013.

[33] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011.

[34] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 1–10, 1985.

[35] M. Luby. Removing randomness in parallel computation without a processor penalty. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 162–173, 1988.

[36] A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu. Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482, 2015.

[37] M. Naor and L. J. Stockmeyer. What can be computed locally? In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 184–193, 1993.

[38] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008.

[39] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

[40] M. Parter. (delta+1) coloring in the congested clique model. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 160:1–160:14, 2018.

[41] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 257–266, 2010.

[42] A. C. Yao. Lower bounds to randomized algorithms for graph properties (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 393–400, 1987.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited