

Dartmouth College

## Dartmouth Digital Commons

---

Dartmouth College Ph.D Dissertations

Theses and Dissertations

---

6-11-2022

# Space-Efficient Algorithms and Verification Schemes for Graph Streams

Prantar Ghosh

Prantar.Ghosh.GR@Dartmouth.edu

Follow this and additional works at: <https://digitalcommons.dartmouth.edu/dissertations>



Part of the [Theory and Algorithms Commons](#)

---

### Recommended Citation

Ghosh, Prantar, "Space-Efficient Algorithms and Verification Schemes for Graph Streams" (2022).

*Dartmouth College Ph.D Dissertations*. 81.

<https://digitalcommons.dartmouth.edu/dissertations/81>

This Thesis (Ph.D.) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Ph.D Dissertations by an authorized administrator of Dartmouth Digital Commons. For more information, please contact [dartmouthdigitalcommons@groups.dartmouth.edu](mailto:dartmouthdigitalcommons@groups.dartmouth.edu).

**SPACE-EFFICIENT ALGORITHMS AND VERIFICATION SCHEMES FOR  
GRAPH STREAMS**

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Prantar Ghosh

Guarini School of Graduate and Advanced Studies

Dartmouth College

Hanover, New Hampshire

May 2022

Examining Committee:

---

Amit Chakrabarti, Chair

---

Deeparnab Chakrabarty

---

Peter Winkler

---

Justin Thaler

---

F. Jon Kull, Ph.D.

Dean of the Guarini School of Graduate and Advanced Studies



# Abstract

Structured data-sets are often easy to represent using graphs. The prevalence of massive data-sets in the modern world gives rise to big graphs such as web graphs, social networks, biological networks, and citation graphs. Most of these graphs keep growing continuously and pose two major challenges in their processing: (a) it is infeasible to store them entirely in the memory of a regular server, and (b) even if stored entirely, it is incredibly inefficient to reread the whole graph every time a new query appears. Thus, a natural approach for efficiently processing and analyzing such graphs is reading them as a *stream* of edge insertions and deletions and maintaining a summary that can be (a) stored in affordable memory (significantly smaller than the input size) and (b) used to detect properties of the original graph. In this thesis, we explore the strengths and limitations of such graph streaming algorithms under three main paradigms: classical or standard streaming, adversarially robust streaming, and streaming verification.

In the *classical* streaming model, an algorithm needs to process an adversarially chosen input stream using space sublinear in the input size and return a desired output at the end of the stream. Here, we study a collection of fundamental directed graph problems like reachability, acyclicity testing, and topological sorting. Our investigation reveals that while most problems are provably hard for general digraphs, they admit efficient algorithms for the special and widely-studied subclass of tournament graphs. Further, we exhibit certain problems that become drastically easier when the stream elements arrive in random order rather than adversarial order, as well as problems that do not get much easier even un-

der this relaxation. Furthermore, we study the graph coloring problem in this model and design color-efficient algorithms using novel parameterizations and establish complexity separations between different versions of the problem.

The classical streaming setting assumes that the entire input stream is fixed by an adversary before the algorithm reads it. Many randomized algorithms in this setting, however, fail when the stream is extended by an adaptive adversary based on past outputs received. This is the so-called *adversarially robust* streaming model. We show that graph coloring is significantly harder in the robust setting than in the classical setting, thus establishing the first such separation for a “natural” problem. We also design a class of efficient robust coloring algorithms using novel techniques.

In classical streaming, many important problems turn out to be “intractable”, i.e., provably impossible to solve in sublinear space. It is then natural to consider an enhanced streaming setting where a space-bounded client outsources the computation to a space-unbounded but untrusted cloud service, who replies with the solution and a supporting “proof” that the client needs to verify. This is called streaming verification or the *annotated streaming* model. It allows algorithms or *verification schemes* for the otherwise intractable problems using both space and proof length sublinear in the input size. We devise efficient schemes that improve upon the state of the art for a variety of fundamental graph problems including triangle counting, maximum matching, topological sorting, maximal independent set, graph connectivity, and shortest paths, as well as for computing frequency-based functions such as distinct items and maximum frequency, which have broad applications in graph streaming. Some of our schemes were conjectured to be impossible, while some others attain smooth and optimal tradeoffs between space and communication costs.

# Acknowledgements

I am thankful to quite a few people in my life, without whom this thesis would not have taken shape. First and foremost, I want to express my gratitude to my advisor Amit Chakrabarti for his excellent guidance and support over the last five years. I fondly cherish the time we have spent brainstorming over problems and discussing ideas. I remember many valuable suggestions that he has given me over the years, not only regarding specific research problems, but about research in general. His mentorship has definitely played a vital role in my growth as a student of computer science, as a researcher, and as a person. I greatly appreciate his patience and dedication towards my academic development, from carefully going through my writing in the earlier days, giving me constructive feedback for any practice talk that I have done, through giving me detailed feedback about my research statement and other application materials in the later days. Above all, I would like to thank Amit for the liberty he has given me for research—the freedom to explore any problem and research direction that I like—and for showing equal enthusiasm for any research problem that I have suggested collaborating on, even if some were not in his primary area of expertise. I have learned a lot from him in the last few years, and I feel very fortunate to have been advised by Amit, a great mentor and a greater human being.

I would like to thank Deeparnab Chakrabarti, Peter Winkler, and Justin Thaler for being on my thesis committee and for their invaluable feedback and suggestions to improve the quality of my thesis. I feel lucky to have had Deeparnab as my friend, philosopher, and guide in the true sense of the words. I have learned a lot from him about research, the

research lifestyle, and the philosophy and the art of teaching. Most importantly, I thank him for being the person with whom I can discuss the most diverse range of topics ranging from research and teaching to sports, movies and TV shows. I am grateful to Pete for all the engaging discussions we have had, and for his ability to make any interaction interesting. I have immensely enjoyed his countless puzzles and the vast collection of intriguing easy-to-explain (but not so easy to solve) problems that he has introduced me to. Above all, I would like to thank Pete for being the wise oracle that I have resorted to, whenever I have encountered conceptual barriers in my research related to combinatorics or probability. I would like to express my sincere gratitude to Justin for being a constant support over the last few years. Of course, I am most thankful to him for making conjectures that I could disprove so as to increase the significance of my results! Jokes apart, I consider myself fortunate to have been able to count on him for multiple aspects including collaboration, feedback on my write-ups, general research and career advice, reference letters, and practically any academic discussion.

I am grateful to Andrew McGregor, Sofya Vorotnikova, Suman Bera, Manuel Stoeckl, and Justin, for their collaborations on the works that contributed to this thesis. None of those works would have been possible without their help, contributions, and sincere engagement. I would also like to thank Sayan Bhattacharya, Jayesh Choudhari, and Sepehr Assadi for their collaborations on other projects that I have thoroughly enjoyed working on during my PhD. I have learned a lot from all of them, and they have all contributed significantly to my academic growth. I would like to specially mention Suman for introducing me to the wonderful area of graph streaming and spurring my initial interest in the field, and for being a mentor and “academic brother” to me in the truest sense.

It is crucial to have a good research environment during one’s PhD life for continuous enrichment and cultivation of one’s ideas and thought process. I would like to thank the members of the theory group at Dartmouth for providing me with the most amazing work

environment: in addition to the group members already mentioned, I thank Hsien-Chih Chang, Prasad Jayanti, Sebastiaan Joosten, Sagar Kale, Maryam Negahbani, Ankita Sarkar, and Hang Liao. I cherish our times in TRG (Theory Reading Group) sessions, a big reason behind my growing interest in research.

Indeed, the life of a PhD student goes beyond just research and lab. I thank everyone who has contributed to my general well-being, my mental health, and social life, which have definitely impacted my work life as well. I am deeply grateful to my friends at Dartmouth who have been a great support system over the last few years: Varun, Ankita, Suman, Maryam, Prashant, Linta, Sisira, Shrea, Saifur, Pradipta, Farzana, Ehsanul, Sharanya, Siddhartha, (Dhananjay) Beri, Shruti, Srivamshi, Priyanshu, and Pushpendra. I thank you for all the memories and all your help and support; I really don't know what I would have done without you. I also thank my friends who have been there for me from hundreds or even thousands of miles away: Soumyajit, Shreejit, Debsuvra, Aneek, Ritwik, (Souvik) Ash, and Sayantan; thanks for being a support through thick and thin, and especially for regular video calls, chats, and online games during the pandemic, when maintaining a good mental health was a challenge. Again, thanks to Sougata, (Suman) Sadhukhan, Ritam, Shiuli, Anirban, Debraj, Arghya, and Rajarshi for all the online board game sessions we have had during the pandemic, which definitely was a mental boost for me. Finally, I thank Janice McCabe and other event organizers at Allen house for all the extra-curricular events that have acted as superb stress relief and I have thoroughly enjoyed them throughout my time at Dartmouth.

Staying abroad for years might be particularly challenging because one does not have family around. I am extremely thankful to Andriela for not having to face this challenge during my PhD life. I can't thank her enough for genuinely making me feel like I have family in the Upper Valley who takes care of me. Thanks to Agastya and Anushka for all the love.

I am grateful to my father Shamik for being my first academic inspiration: he is the first reason for my interest in math, for me wanting to pursue a PhD, and for my choice of a career path that I never regret. Finally, I would like to thank my mother Nandita for literally everything. I dedicate my thesis to her and hope that it makes a convincing case for why staying away for all these years was kinda worth it.

# Contents

|  |           |
|--|-----------|
| Abstract . . . . .   | ii        |
| Acknowledgements . . . . .   | vii       |
| <b>1 Introduction</b>  | <b>1</b>  |
| 1.1 Overview of Results and Contributions . . . . .                      | 7         |
| 1.1.1 The Classical Streaming Model . . . . .                            | 7         |
| 1.1.2 The Adversarially Robust Streaming Model . . . . .                 | 10        |
| 1.1.3 The Annotated Streaming Model . . . . .                            | 11        |
| 1.2 Standard Techniques . . . . .  | 15        |
| 1.2.1 Sketching . . . . .  | 15        |
| 1.2.2 Communication Complexity . . . . .                                 | 17        |
| 1.3 Notations, Terminology, and Basic Tools . . . . .                    | 19        |
| <b>2 Classical Graph Streaming</b>                                       | <b>21</b> |
| 2.1 Directed Graph Problems . . . . .                                    | 23        |
| 2.1.1 Our Results . . . . .  | 24        |
| 2.1.2 Previous Work . . . . .  | 27        |
| 2.1.3 Problems and Preliminaries . . . . .                               | 28        |
| 2.1.4 General Digraphs and the Hardness of Some Basic Problems . . . . . | 30        |
| 2.1.5 Sink Finding in Tournaments . . . . .                              | 40        |
| 2.1.6 Feedback Arc Set in Tournaments . . . . .                          | 47        |

|          |   |            |
|----------|---|------------|
| 2.1.7    | Topological Ordering in Random Graphs . . . . .                   | 56         |
| 2.1.8    | Rank Aggregation . . . . .  | 61         |
| 2.1.9    | Subsequent Works . . . . .  | 62         |
| 2.2      | Graph Coloring . . . . .  | 63         |
| 2.2.1    | Our Results and Techniques . . . . .                              | 65         |
| 2.2.2    | Related Work and Comparisons . . . . .                            | 69         |
| 2.2.3    | Preliminary tools . . . . .                                       | 75         |
| 2.2.4    | LDP: A Generic Framework for Coloring . . . . .                   | 77         |
| 2.2.5    | Streaming Algorithm for Degeneracy-Based Coloring . . . . .       | 81         |
| 2.2.6    | Streaming Lower Bounds . . . . .                                  | 84         |
| 2.2.7    | Applications in Various Space-Conscious Models . . . . .          | 92         |
| 2.2.8    | A Combinatorial Lower Bound . . . . .                             | 104        |
| 2.2.9    | Subsequent works . . . . .  | 107        |
| <b>3</b> | <b>Adversarially Robust Streaming</b>                             | <b>109</b> |
| 3.1      | Motivation and Context . . . . .                                  | 111        |
| 3.2      | Adversarially Robust Coloring . . . . .                           | 117        |
| 3.2.1    | Our Results and Contributions . . . . .                           | 117        |
| 3.2.2    | Preliminaries . . . . .   | 119        |
| 3.2.3    | Overview of Techniques . . . . .                                  | 123        |
| 3.2.4    | Hardness of Adversarially Robust Graph Coloring . . . . .         | 129        |
| 3.2.5    | Upper Bounds: Adversarially Robust Coloring Algorithms . . . . .  | 136        |
| 3.2.6    | An Algorithm Based on Palette Sparsification . . . . .            | 137        |
| 3.2.7    | Sketch-Switching Based Algorithms for Turnstile Streams . . . . . | 144        |
| <b>4</b> | <b>Streaming Verification</b>                                     | <b>160</b> |
| 4.1      | Preliminaries, Setup, and Terminology . . . . .                   | 161        |

|          |  |            |
|----------|--|------------|
| 4.2      | Frequency-Based Functions . . . . .                              | 164        |
| 4.2.1    | Our Results and Techniques . . . . .                             | 166        |
| 4.2.2    | The Misra-Gries Algorithm . . . . .                              | 170        |
| 4.2.3    | Computing Frequency-based Functions in Turnstile Streams . . . . | 171        |
| 4.2.4    | Modifications for Longer Streams . . . . .                       | 178        |
| 4.3      | Graph Problems . . . . .   | 182        |
| 4.3.1    | Our Techniques . . . . .   | 184        |
| 4.3.2    | Triangle Counting . . . . .                                      | 187        |
| 4.3.3    | Generalization: Counting Copies of an Arbitrary Subgraph . . . . | 193        |
| 4.3.4    | A Technical Result: Counting Edges in Induced Subgraphs . . . .  | 195        |
| 4.3.5    | Maximum Matching . . . . .                                       | 198        |
| 4.3.6    | Applications to Other Graph Problems . . . . .                   | 203        |
| 4.3.7    | Path Problems . . . . .  | 207        |
| 4.4      | Multipass Stream Verification . . . . .                          | 217        |
| 4.4.1    | One-Pass Lower Bounds . . . . .                                  | 217        |
| 4.4.2    | Two-pass Scheme for CROSSEGECOUNT with Applications . . .        | 218        |
| 4.4.3    | A Multi-Pass Scheme for Detecting Short Paths . . . . .          | 224        |
| <b>5</b> | <b>Conclusions and Future Directions</b>                         | <b>227</b> |
|          | <b>References</b>  | <b>233</b> |

---

# Chapter 1

---

## Introduction

Big data is ubiquitous in the modern world. The underlying structure of a data-set often conforms to a natural *graph representation*. For instance, the users of a social medium and their interactions or connections are easily modeled by the social network graph: make each user a node and include an edge between two users if and only if they connect or interact on the social medium. The vast growth of big data has given rise to *big graphs* or *massive graphs* such as *social networks*, *web graphs*, *citation networks*, *transportation networks*, *communication networks*, *collaboration networks*, and *biological networks* such as *protein* and *brain networks*. Concrete examples of such real-world graphs include the *Facebook graph* containing around a trillion edges [67], the *English Wikipedia graph* containing more than 150 million edges<sup>1</sup>, and the *patent citation network* of citations made by U.S. patents between 1975 and 1999 containing more than 16.5 million edges [129] (see SNAP datasets [130] for more examples of such large real-world networks). To this end, *big data analysis* significantly involves *big graph processing*. The big challenge in big graph processing is that the size of the input is strikingly larger than the memory of a regular processor. Further, most of these graphs keep growing over time, making the storage of the entire input even more infeasible. Even if we can afford the memory to store the whole graph, it is incredibly

---

<sup>1</sup><https://law.di.unimi.it/webdata/enwiki-2021/>

inefficient to reread the entire input every time a new query appears or for *real-time analysis* when edges are getting inserted and deleted rapidly; think of a viral post proliferating on social media and a large number of its copies getting deleted upon being detected as *fake news*. Hence, a natural approach for processing massive graphs is to process them part by part. This leads to the idea of reading the input as a *stream*: an algorithm receives the input graph as a sequence of edge insertions or deletions and maintains a summary using the available memory (which is significantly smaller than the input size). At the end of the stream, it answers certain queries about the input based on the stored summary. This is called the *graph streaming* model which we explore in this thesis. Our goal is to design space-efficient algorithms in this setting for fundamental graph problems and prove their limitations, e.g., the minimum space required by any streaming algorithm for a certain problem. Apart from the *classical* or *standard* streaming setting described above, we also study two of its variants, namely *adversarially robust* streaming and *annotated* streaming or streaming *verification*.

**Classical Streaming.** In the standard streaming setting, we lay the foundation for directed graph algorithms. Observe that a large number of social media graphs, e.g., *follows* on *Twitter/Instagram* and other real-world big graphs such as citation networks and web graphs with *hyperlinks* are directed graphs. However, prior work on graph streaming focused almost exclusively on undirected graphs. We consider fundamental digraph problems such as *topological sorting*, *feedback arc set*, testing *acyclicity* and *reachability*, and finding *source/sink* nodes. For practical motivation behind these problems, think of fake news spreading rapidly over social media, and we seek to find its source. Or, we find a topological ordering of the nodes of a citation network (which is acyclic assuming that a paper only cites past papers) to obtain a chronological order of the papers in the network. In a nutshell, our results show that although most of these problems turn out to be “hard” in streaming, they do admit efficient algorithms for the special class of *tournament graphs*

(digraphs with exactly one arc between each pair of nodes). We consider the case where the stream-arrival order is adversarial as well as the case where it is random. We exhibit problems of two categories: (i) ones that are roughly equally hard for both types of stream orders and (ii) ones that need exponentially larger space for adversarial order as opposed to random order.

Now, note that all digraph problems that we study deal with *vertex orderings*. We demonstrate that vertex orderings can be useful even for undirected graphs as they yield a simple analysis for a streaming algorithm that we design for a fundamental undirected graph problem, namely the graph coloring problem. Here, we need to assign colors to the nodes of a graph such that the end-points of each edge receive different colors. In real-world big graphs such as social networks, a coloring can be used to detect *community structures* [143]: observe that a vertex coloring partitions the graph into independent sets since each color induces such a set. The presence of large independent sets or cliques (which are independent sets in the complement graphs) provides crucial information about the social network. Our algorithm uses significantly fewer colors than the state of the art for most graphs including real-world graphs and sparse graphs while being much simpler at the same time. Further, the general framework of the algorithm can be implemented in multiple other “space-conscious” settings to get improved graph coloring algorithms. To complement our algorithmic results, we show that any algorithm that significantly improves on our color bound must store almost the entire graph. We give a summary of our main contributions and results in the standard streaming model in Section 1.1.1; the full details appear in Chapter 2.

**Adversarially Robust Streaming.** The classical streaming setting models worst case inputs by having an adversary fix a hard input stream before the algorithm sees it. Thus, the adversary cannot decide the upcoming stream elements on the fly based on the current output of the algorithm. Consider, however, an online marketing service like *Amazon* running

an algorithm that processes the Amazon *product-co-purchasing network* [128], which has products as nodes and edges between commonly co-purchased products, and recommends items to users based on some statistics of the graph. Then, the next purchases of the user, and hence, the next edges in the said network, are heavily dependent on the past outputs (recommendations) of the algorithm. Hence, such a scenario cannot be modeled by the classical streaming setting. To overcome this, consider a streaming model with a more powerful *adaptive* adversary who, at every point in the stream, can query the algorithm, receive an output, and decide the next stream token based on all past outputs and stream elements. This model does capture the worst case inputs in the above scenario and other practical scenarios where the adaptivity can indeed be more “adversarial”: suppose that Alice looks at the stream of orders in a high frequency stock market and runs an algorithm on the stream to decide her own order, while her competitor Charlie observes her orders and buys stocks so as to tamper with the input stream and mislead Alice’s algorithm [97]. It is natural to seek streaming algorithms that work even against such an adaptive adversary. We call such an algorithm *adversarially robust*. Indeed, deterministic streaming algorithms are always adversarially robust. A randomized algorithm that works for classical streaming, however, may not be robust: an adaptive adversary can *learn* the algorithm’s random bits and extend the stream in such a way that the final stream turns out to be one on which those random bits fail. On the other hand, the adversary in classical streaming that fixes the input in advance is *oblivious* to the random bits used by the algorithm. Thus, the contrast between the classical and the robust streaming settings can be simply seen as oblivious adversary versus adaptive adversary, a well-known concept in various other settings such as the *online* and *dynamic* models. To establish a separation between robust and classical streaming, prior work exhibited a fairly artificial problem that is much harder in the former setting than in the latter. In this thesis, we show that graph coloring is significantly harder against an adaptive adversary than an oblivious one, thus establishing the first

such separation for a natural and well-studied problem. Further, it exhibits the first separation between deterministic and randomized streaming algorithms for graph coloring since any deterministic algorithm must be robust. Whether either of these separations could be shown were major open questions. Furthermore, we design adversarially robust coloring algorithms using a reasonably small number of colors. One significance of our algorithms is that, combined with results from our work as well other ones in the literature, they show a double-separation for streaming graph coloring: for the optimal space regime, a robust algorithm requires notably more colors than ordinary randomized algorithms, but significantly fewer colors than deterministic algorithms. We state our results and contributions in more detail in Section 1.1.2, and give the full details in Chapter 3.

**Streaming Verification.** It can be proven that many natural problems in the classical streaming setting require storage of almost the entire graph. To surmount this, consider the following modification of the streaming model motivated by cloud-computing: a space-bounded client reading a huge input stream outsources the computation to a cloud service with unbounded space. The cloud returns the desired solution to the client who, however, refuses to blindly trust it. They fear that the cloud might have encountered some bug, incurred some hardware or network failure, or might be malicious or compromised, leading to a corrupted solution. Hence, they ask the cloud for a *proof* to justify the solution. The client (now called Verifier) then uses the information that they extracted from the input stream on their own to *verify* the proof and the solution sent by the cloud (now called Prover). This model combining Prover-Verifier systems with data streaming was introduced by Chakrabarti, Cormode, and McGregor [55] as the *annotated streaming* model. A protocol in this model, called a *verification scheme* or simply *scheme*, aims to optimize the space used for verification and the number of bits used to express the proof. It turns out that a large number of fundamental problems that are not solvable in sublinear space in classical streaming *do* admit schemes with both verification space and proof length sublinear in the

input size. In this thesis, we design efficient schemes for a variety of graph problems as well as for computing frequency-based functions that are important primitives for graph streaming.

Since its inception, data streaming has mostly dealt with statistical problems involving item frequencies. Consider a very general problem: for any given function, we need to compute the sum of the functional values of the stream frequencies. Special cases include fundamental data stream problems such as computing the number of *distinct items* ( $F_0$ ) in the stream, the *frequency moments* ( $F_k$ ), and *heavy-hitters*. It can be also applied to calculate the maximum frequency of an item ( $F_\infty$ ). Such *frequency-based functions* have wide applications in graph streaming, e.g.,  $F_0$  can be used to count the number of distinct edges in a multigraph (or a subgraph of it) and frequency moments  $F_k$  for small  $k$  have been used to design efficient algorithms for triangle counting [25]. In the annotated setting, they have been used, for instance, to design protocols for maximum matching [59] and checking graph connectivity [57]. Thus, apart from being of independent interest, efficient protocols for frequency-based functions would imply more efficient subroutines and protocols for graphs streaming problems. We describe a scheme for computing general frequency-based functions which is significantly simpler and slightly more efficient than the previously best-known one. It also implies improved and simpler schemes for the special cases of computing  $F_0$  and  $F_\infty$ .

The most extensively-studied graph problems in streaming are *triangle counting*, its generalization to *subgraph counting*, and *maximum matching*. The subgraph counting problem has numerous applications in *data mining* and *large network analysis*. In particular, triangle counting has found use in *spam detection* [37], *motif discovery in protein networks* [117], and in several measures used for community structure analysis in social networks such as *transitivity* [167] and *clustering coefficients* [145]. The maximum matching problem has been applied in *ad allocation* on the internet. We give efficient schemes

for each of these problems. Some of these schemes were conjectured to be impossible, while some others exploit the tool of *non-linear sketches*, as opposed to the standard linear sketches, to attain optimal schemes with smooth tradeoffs between the space and proof length. Further, we design general frameworks that can be applied to obtain efficient schemes for a variety of graph problems such as *topological sorting*, *acyclicity testing*, *maximal independent set*, and *graph connectivity*. We also devise new schemes for a class of *path problems*. Furthermore, we introduce the *multi-pass* annotated streaming model (analogous to classical multi-pass streaming), where the verifier can make multiple passes over the input stream before receiving the proof from the Prover. We show that some problems become provably easier when two passes are allowed rather than a single pass. We also develop efficient multi-pass schemes for certain problems. We discuss the main results and contributions more formally in Section 1.1.3 and then present the full details in Chapter 4.

## Section 1.1

### Overview of Results and Contributions

In this section, we describe our main results and contributions in each variant of the streaming model that we study.

#### 1.1.1. The Classical Streaming Model

We briefly describe our results for classical graph streaming here. The details appear in Chapter 2. In Section 2.1, we present our results on directed graph streams, based on a joint work with A. Chakrabarti, A. McGregor, and S. Vorotnikova [60]. Next, in Section 2.2, we give an account of our results on graph coloring from a paper with A. Chakrabarti and S.K. Bera [43].

**Digraph problems.** Directed graphs had not received much attention in the data streaming

community until our work [60] laid the foundation for their study. Consider the problem of *topological sorting*, where, given a digraph, we need to find an ordering of the nodes such that all edges go “forward”. This problem, probably the most classical problem exclusive to directed graphs, has offline algorithms known for more than half a century. But its streaming complexity was open prior to our work. We prove that topological sorting does not admit any sublinear-space algorithm in a single pass (Theorem 2.1.3). Moreover, for  $p$  passes on an  $n$ -vertex graph, it requires roughly  $\Omega(n^{1+1/p})$  space (Corollary 2.1.5). We show that this bound also applies to related problems such as testing *acyclicity* (Proposition 2.1.4) and *feedback arc set* (Corollary 2.1.6). Similar lower bounds were previously known for testing *node-to-node reachability* [95]. However, all these lower bounds crucially utilized the fact that the stream is adversarially ordered. Could it be that these problems become easier if the edges arrive in a random order? Our main result in this section (Theorem 2.1.9) answers this question in the negative, thus establishing the first lower bound that precludes semi-streaming space (i.e.,  $O(n \text{ polylog}(n))$  space) for some graph problem on random-order streams. This lower bound also applies for widely studied undirected graph problems such as detecting whether a graph has a perfect matching or a short  $s$ - $t$  path (Corollary 2.1.10).

On the other hand, for the problem of finding a sink (or equivalently, source) node in a tournament graph (where each pair of vertices shares exactly one directed edge), we show an exponential separation in space complexity between adversarial- and random-order streaming (Theorems 2.1.14 and 2.1.15). Further, we demonstrate that the special class of tournaments allows interesting algorithms for many of the aforementioned problems. In particular, for feedback arc set in tournaments (FAST), we give two semi-streaming algorithms: a one-pass  $(1 + \varepsilon)$ -approximation algorithm with exponential post-processing time (Theorem 2.1.19), and a poly-time  $\log n$ -pass 3-approximation algorithm (Theorem 2.1.22). We also prove a lower bound that implies that an exponential post-processing time is necessary for *any* algorithm that uses the natural sketching technique

that we use (Theorem 2.1.29). Furthermore, for random DAGs over certain natural well-studied distributions, we showed that there exist efficient sublinear space algorithms for topological sorting (Theorems 2.1.33 and 2.1.34). This implies that the standard technique, which establishes randomized lower bounds by proving that a problem is hard for such distributions over the inputs, does not work for the topological sorting problem. To obtain such a lower bound, we need to come up with some other involved distributions or more sophisticated techniques. Finally, we exhibit that our algorithmic techniques can be used to solve *rank aggregation* (Theorem 2.1.35), a widely studied problem in practice.

**The vertex-coloring problem.** We turn to the *vertex-coloring* problem, another fundamental graph problem in theoretical computer science. The task is to color the vertices of a graph such that no two vertices sharing an edge receive the same color. Given the hardness of coloring with the minimum number of colors, a long line of research has focused on  $(\Delta + 1)$ -coloring a graph, where  $\Delta$  is its maximum vertex degree. The offline greedy  $(\Delta + 1)$ -coloring algorithm takes linear space, which is infeasible for massive graphs. A breakthrough result by Assadi, Chen, and Khanna [17] gave a  $(\Delta + 1)$ -coloring semi-streaming algorithm. However, a  $(\Delta + 1)$ -coloring is often wasteful, especially for certain sparse graphs (for instance, think of a star graph). In light of this, we consider coloring with respect to a “better” parameter called the *degeneracy*  $\kappa$  of the graph: every graph is  $(\kappa + 1)$ -colorable and  $\kappa$  is always at most  $\Delta$ . In fact, it can be arbitrarily smaller than  $\Delta$  (1 vs  $n - 1$  for star graphs). We design a semi-streaming algorithm that colors every graph with  $\kappa(1 + o(1))$  colors; the algorithm is much simpler and more color-efficient than previous work for most sparse graphs and real-world massive graphs. An important feature of our algorithm is that it can be implemented in multiple other space-conscious settings such as graph query and certain distributed models, improving the state of the art for each of these settings. Moreover, it attains fewer colors with simpler analysis even compared to *arboricity-based coloring*, which is a color-saving regime more popular among prior

works. Thus, the result also conveys a notable conceptual message: for graph coloring in multiple settings, degeneracy is a better parameter than the more widely-studied arboricity.

Can we improve the number of colors all the way down to the combinatorially optimal  $\kappa + 1$ ? We show lower bounds answering this in the negative: an exact  $(\kappa + 1)$ -coloring algorithm cannot achieve sublinear space. In general, we achieve a smooth tradeoff between the number of colors and the space required, implying that a semi-streaming algorithm must use at least  $\kappa + \sqrt{\kappa}$  colors, justifying the super-constant additive slack in our upper bound. This result conveys yet another important conceptual message: there is a large gap between the space requirements for  $\Delta + 1$  and  $\kappa + 1$  colorings even though the two parameters are somewhat analogous. For a detailed discussion on the results, see Section 2.2.1.

### 1.1.2. The Adversarially Robust Streaming Model

Here, we briefly describe the results presented in Chapter 3, which are from a joint work with A. Chakrabarti and M. Stoeckl [61]. Recall that Assadi, Chen, and Khanna [17] gave a semi-streaming algorithm using  $\Delta + 1$  colors in the classical streaming model (where the adversary is oblivious). We observe that an adaptive adversary can not only break this algorithm, but all previously known streaming coloring algorithms. This is because all these algorithms are randomized: the adversary can *learn* the random bits used by such an algorithm from its outputs and then extend the stream to one that is bad for those random bits. In Section 3.2.4, we formally prove that coloring is indeed harder in the robust setting: (a) any robust  $O(\Delta)$ -coloring algorithm must use  $\Omega(n\Delta)$ , i.e., linear space, and (b) any robust semi-streaming coloring algorithm must use at least  $\Omega(\Delta^2)$  colors. Contrasting with the aforementioned result of Assadi, Chen, and Khanna [17], these results resolve two important open questions in streaming raised by past work: (i) whether there is a *natural* problem that is significantly harder for adversarially robust streaming than classical streaming, and (ii) whether deterministic streaming algorithms for graph coloring require much more space than randomized ones. The latter follows from the fact that determinis-

tic algorithms are always robust. We establish the lower bound using the standard tool of communication complexity; however, we reduce from a novel communication game called *subset avoidance* which might be of independent interest. Further, our reduction uses innovative techniques tailored to show robust streaming lower bounds as opposed to standard streaming ones.

For the robust coloring problem, the biggest challenge seems to be the following: while the adaptive adversary can force us to change our output at every step (by introducing an edge between two like-colored nodes in the current output), prior algorithmic techniques in the robust streaming literature work only for problems where an algorithm can be forced to change its output only a small number of times over the course of the stream. We come up with new techniques to overcome this challenge and design a class of robust coloring algorithms using  $\text{poly}(\Delta)$  colors. To be precise, we obtain an  $O(\Delta^2)$ -coloring in  $O(n\sqrt{\Delta} \cdot \text{polylog}(n))$  space, and an  $O(\Delta^3)$ -coloring in semi-streaming space. Contrasting the latter with a result of Assadi, Chen, and Sun [16] that shows that a deterministic semi-streaming coloring algorithm must use  $\exp(\Delta^{\Omega(1)})$  colors, we get a separation between robust and deterministic streaming. Thus, together with the gap shown by our lower bounds, we get the first double-separation between ordinary randomized, adversarially robust, and deterministic streaming.

### 1.1.3. The Annotated Streaming Model

We summarize our results and contributions on streaming verification here. The details appear in Chapter 4. In Section 4.2, we present the author's work [89] on frequency-based functions. The material in Section 4.3 is based on two papers: a joint work with A. Chakrabarti [59] and another with A. Chakrabarti and J. Thaler [62].

We first define some terminology for ease of presentation of the results. For an annotated streaming scheme, we call the number of bits used to express Prover's help message or proof as *hcost* of the scheme and the number of bits of space used by Verifier as *vcost*

of the scheme. An  $(h, v)$ -scheme (resp.  $[h, v]$ -scheme) denotes a scheme with  $O(h)$  (resp.  $O(h \log n)$ ) hcost and  $O(v)$  (resp.  $O(v \log n)$ ) vcost.

**Frequency-based functions.** Given a stream with elements in universe  $\{1, \dots, n\}$ , let  $\mathbf{f}$  denote its frequency vector  $\langle f_1, f_2, \dots, f_n \rangle$ , where  $f_j$  is the frequency (can be negative in case of turnstile streams) of the  $j$ th element. A *frequency-based function* is a function  $G$  where  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  for some integer-valued function  $g$ . Setting  $g$  accordingly evaluates  $G(\mathbf{f})$  as the answer to fundamental data stream problems such as the number of distinct items ( $F_0$ ),  $k$ th frequency moment ( $F_k$ ), or number of items with frequency above a certain threshold (*heavy hitters*). For any turnstile stream of length  $m = O(n)$ , Chakrabarti et al. [57] designed an  $[n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n]$ -scheme that computes  $G(\mathbf{f})$  for any given function  $g$ . Their scheme uses an intricate data structure with binary trees and calls upon a subroutine for heavy hitters that uses an elaborate framework called *hierarchical heavy hitters*. Given how general the problem is, with several special cases having numerous applications, it is important and beneficial to have a simple scheme for the problem. In Section 4.2, we design such a simple scheme that uses the most basic and classical frequency estimation data structure for heavy-hitters: the Misra-Gries summary [141]. At the same time, the scheme improves upon [57]’s complexity bounds: it is an  $[n^{2/3} \log n, n^{2/3} \log n]$ -scheme. No scheme better than the direct application of the general one was known even for the special cases of computing  $F_0$  or  $F_\infty$ . Our result thus simplifies and improves the bounds for these important problems as well.

This scheme using Misra-Gries works only for stream length  $m = O(n)$ . Although Chakrabarti et al. [57] had made the same assumption, their scheme can be made to work for longer streams as long as the sum of the frequencies  $\|\mathbf{f}\|_1$  is  $O(n)$ . We show that we can modify our scheme to handle such streams while preserving the same complexity bounds, albeit at the cost of imperfect completeness. The modification involves replacing the Misra-Gries subroutine by the Count-Median sketch [72] that gives stronger guarantees at the cost

of randomization.

**Graph problems.** Two graph problems that we largely focus on in the annotated streaming setting are *triangle counting* and *maximum matching*. For both of these problems on  $n$ -vertex graphs, Thaler [164] gave semi-streaming schemes, i.e.,  $[n, n]$ -schemes, which match (up to polylogarithmic factors) a lower bound given by Chakrabarti et al. [57] that says that any  $(h, v)$ -scheme for these problems requires  $hv \geq n^2$ . However, even the number of vertices  $n$  might be too large for the verifier to afford  $\tilde{O}(n)$  bits of space. Hence, a natural question was whether a  $o(n)$ -space protocol could be achieved with sublinear, i.e.,  $o(n^2)$  proof length. Thaler [164] conjectured that such a scheme does not exist for either the triangle counting or the maximum matching problem. Our main result in this work disproved the conjecture giving  $(o(n^2), o(n))$ -schemes for both of these problems. Furthermore, for counting general subgraphs of constant size  $k > 3$ , we designed the first sublinear, i.e.,  $(o(n^2), o(n^2))$ -scheme.

Towards the goal of fully settling the complexity of these problems in the annotated setting, we attempt to match the aforementioned lower bound by obtaining  $[h, v]$ -schemes with  $hv = n^2$  for *all* possible settings of  $h$  and  $v$ . This means we want to get smooth tradeoffs between  $h$  and  $v$  over the entire curve  $hv = n^2$ . Our first class of schemes for both problems are  $[t^3, s^2]$ -schemes for any  $t, s$  satisfying  $ts = n$ . They do attain a smooth tradeoff but clearly do not match the lower bound. Thaler's [164]  $[n, n]$ -schemes did match the lower bound, but they did not achieve a smooth tradeoff. In fact, on the optimal tradeoff curve  $hv = n^2$ , schemes were known only for the settings  $(h = n^2, v = 1)$  [57],  $(h = n, v = n)$  [164], and the trivial  $(h = 1, v = n^2)$ . We combine the techniques used in our protocol for with those in Thaler's protocol and succeed in achieving the best of both of worlds: we obtain smooth and optimal tradeoffs for the problems that cover a significant portion of the curve. Specifically, for triangle counting and maximum matching, we gave  $[h, v]$ -schemes for any  $h, v$  with  $hv = n^2$ , provided  $h \leq n$  and  $h \geq n$  respectively. Thus,

these schemes settle the complexity of the triangle counting and the maximum matching problems in the *laconic* (i.e.,  $(\text{hcost} \leq n, \text{vcost} \geq n)$ ) regime and the *frugal* (i.e.,  $(\text{hcost} \geq n, \text{vcost} \leq n)$ ) regime respectively. Furthermore, for triangle counting, we design an improved  $[nt^2, s]$ -scheme for any  $t, s$  with  $ts = n$ .

To solve the maximum matching problem, we use a framework called *Induced-Edge-Count* that counts the total number of edges contained in a given collection of vertex-subsets. We show that this framework can be also applied to get optimal schemes with smooth tradeoffs for several well-studied problems in the streaming model, namely *maximal independent set*, *topological sorting*, and *acyclicity testing*. For any  $t, s$  satisfying  $ts = n$ , we obtain  $[nt, s]$ -schemes for each of these problems. Note that the first two problems have output size  $\Theta(n)$ . Hence, a scheme with  $o(n)$  space for these problems means that the prover streams the solution to the verifier who verifies it using only  $o(n)$  space. We also use it to design efficient schemes for triangle counting in sparse graphs and in the vertex-arrival/adjacency-list model.

We explore a collection of path problems in this model. We give a  $[kn, n]$ -scheme for the (unweighted) *shortest  $s$ - $t$  path* problem, where we need to find the length  $k$  of the shortest path from node  $v_s$  to node  $v_t$ . Our scheme is optimal when  $k$  is polylogarithmic in  $n$  and also improves upon the proof length of a previous scheme given by Cormode et al. [71]. The more general *single-source shortest path* (SSSP) problem asks for the distances from a source node  $v_s$  to all other nodes in the graph. For any  $t, s$  with  $ts = n$ , we design a  $[Dnt, s]$ -scheme for the unweighted SSSP problem (where  $D$  is the maximum distance from the source vertex to any vertex reachable from it), which can be adapted to a  $[knt, s]$ -scheme for unweighted shortest  $v_s$ - $v_t$  path (where  $k$  is the length of such a path). The latter result strictly improves upon Cormode et al.'s  $[Dnt, s]$ -scheme [71] for the problem (since  $k \leq D$  always). We also have some results for the weighted SSSP problem that are optimal for polylogarithmic weights and diameter.

Next, we introduce the concept of *multi-pass* schemes. Here, analogous to classical streaming, Verifier can make several passes over the input stream before he receives the proof. We showed that some natural problems admit provably better schemes when additional passes are allowed. For instance, the *independent set testing* problem, where we need to determine whether a streamed subset of vertices (arbitrarily interleaved with the edges) form an independent set in the graph, requires total cost  $h + v = \Omega(n)$  for any single-pass  $(h, v)$ -scheme. In contrast, we designed a 2-pass scheme with total cost  $\tilde{O}(n^{2/3})$  for the problem. In fact, we obtained a smooth tradeoff by designing a  $[t^2, s]$ -scheme for any  $t$  and  $s$  with  $ts = n$ . Finally, for the *st-kPath* problem that asks whether there exists a path of length at most  $k$  between nodes  $v_s$  and  $v_t$ , we showed that one can break the  $\Omega(n)$  barrier at the cost of a few passes. To be precise, we gave a  $\lceil k/2 \rceil$ -pass  $[n^{1-1/k}, n^{1-1/k}]$ -scheme for the problem.

## Section 1.2

### Standard Techniques

Almost all our streaming algorithms use *sketching* in some way, and all our streaming lower bounds are proven via communication complexity. In this section, we discuss these two broad techniques that we use throughout the thesis.

#### 1.2.1. Sketching

To obtain sublinear space streaming algorithms, it is clear that instead of storing the entire input, we need to maintain a summary or *sketch* of it while reading the stream. In the context of streaming algorithms, the word “sketch” means more: a data structure is called a sketch if its contents for two different streams  $\sigma_1$  and  $\sigma_2$  can be efficiently combined to obtain what it would store when processing the union of the two streams  $\sigma_1 \circ \sigma_2$ .

**Definition 1.2.1** (Sketch). A data structure  $\mathcal{S}(\sigma)$  computed when an algorithm  $\mathcal{S}$  processes

a stream  $\sigma$  is called a *sketch* if there exists a space-efficient combining algorithm  $\mathcal{A}$  such that, for any two streams  $\sigma_1$  and  $\sigma_2$  (where  $\sigma_1 \circ \sigma_2$  is a valid input stream for  $\mathcal{S}$ ), we have  $\mathcal{A}(S(\sigma_1), S(\sigma_2)) = S(\sigma_1 \circ \sigma_2)$ .

Suppose that our stream arrives from a universe  $\{1, \dots, N\}$ . Thus, a stream of length  $m$  is an element in  $\{1, \dots, N\}^m$ . Denote the frequency vector of the stream  $\sigma$  by  $\mathbf{f}_\sigma$ . For problems on  $n$ -node graphs, we can have some canonical indexing of the edges and take  $N = \binom{n}{2}$ . Then for simple graphs, the frequency vector is an  $N$ -length binary string. A sketch processing stream  $\sigma$  can be seen as maintaining a function of  $\mathbf{f}_\sigma$ . A particular case of interest is when it maintains a *linear* function of  $\mathbf{f}_\sigma$ . We call such a sketch to be a *linear sketch*.

**Definition 1.2.2** (Linear Sketch). The sketch  $\mathcal{S}(\sigma)$  maintained by an algorithm  $\mathcal{S}$  processing stream  $\sigma$ , where  $\mathcal{S}(\sigma) =: \mathcal{S}'(\mathbf{f}_\sigma) \in X$  for some vector space  $X$ , is called a *linear sketch* if for any two streams  $\sigma_1$  and  $\sigma_2$  (where  $\sigma_1 \circ \sigma_2$  is a valid input stream for  $\mathcal{S}$ ), we have  $\mathcal{S}'(\mathbf{f}_{\sigma_1} + \mathbf{f}_{\sigma_2}) = \mathcal{S}'(\mathbf{f}_{\sigma_1}) + \mathcal{S}'(\mathbf{f}_{\sigma_2})$ , i.e.,  $\mathcal{S}(\sigma_1 \circ \sigma_2) = \mathcal{S}(\sigma_1) + \mathcal{S}(\sigma_2)$ .

Observe that for linear sketches, the combining algorithm mentioned in Definition 1.2.1 simply adds the sketches (in the underlying vector space). In fact, upon receiving a new stream token  $j$ , it simply updates the sketch by adding  $\mathcal{S}(\langle j \rangle)$  to the current vector.

A linear sketch  $\mathcal{S}$  processing a stream on universe  $\{1, \dots, N\}$  using space  $s$  (roughly) is popularly interpreted as left multiplication by a matrix in  $\mathbb{R}^{s \times N}$ . Assume that the sketch  $\mathcal{S}$  maintains a vector  $v \in \mathbb{R}^s$  while processing the stream. Then, we have  $\mathcal{S}(\sigma) = S\mathbf{f}_\sigma = v$  for a suitable sketch matrix  $S \in \mathbb{R}^{s \times N}$ . Thus,  $\mathcal{S}(\langle j \rangle)$  that is added upon arrival of token  $j$  is simply the  $j$ th column of the matrix  $S$ . Indeed, a sketching algorithm performs the multiplication implicitly: the matrix  $S$  cannot be stored explicitly in sublinear space.

We use linear sketching in many of our algorithms. In Chapter 4, though, we heavily use *non-linear sketches* and demonstrate how they turn out to be crucial in designing efficient verification schemes.

### 1.2.2. Communication Complexity

Communication complexity is a standard tool used in the streaming literature to show space lower bounds. It also has wide applications beyond streaming. Across all the chapters in this thesis, we use results and techniques in communication complexity to establish our streaming lower bounds. Here, we give a brief introduction to the basic tools that are relevant to this thesis. For details, we refer the reader to the book by Kushilevitz and Nisan [125].

**Two-party communication model.** The two-party communication model introduced by Yao [170] is as follows. The two parties or players Alice and Bob possess inputs  $a \in A$  and  $b \in B$  respectively, for some arbitrary sets  $A$  and  $B$ . Given some relation (equivalently, a problem)  $\mathcal{P} \subseteq A \times B \times C$  for some set  $C$ , Alice and Bob need to compute  $c \in C$  such that  $(a, b, c) \in \mathcal{P}$ . For this purpose, they decide on some communication protocol  $\Pi$ . The protocol  $\Pi$  proceeds in rounds: it determines the player in the first round to send a message to the other player, who looks at the message and replies according to  $\Pi$  in the second round, and then they go back and forth. After some rounds of communication, the protocol  $\Pi$  terminates when one of the players announces an output denoted by  $\Pi(a, b)$ . The *communication cost* of a protocol is measured by the total number of bits communicated over all the rounds in the worst case (assume that Alice and Bob are computationally unbounded, and so no other complexity parameters are taken into account). Formally, denote the total number of bits communicated in protocol  $\Pi$  on inputs  $a$  and  $b$  by  $\text{CC}_{a,b}(\Pi)$ . Then, the communication cost of the protocol  $\Pi$  is given by  $\text{CC}(\Pi) := \max_{(a,b) \in A \times B} \text{CC}_{a,b}(\Pi)$ .

If for any pair of inputs  $(a, b)$ , the output  $\Pi(a, b)$  is always correct, i.e., if  $(a, b, \Pi(a, b)) \in \mathcal{P}$  always, then we say that the protocol  $\Pi$  *deterministically solves*  $\mathcal{P}$ . We are now ready to define the deterministic communication complexity of a problem.

**Definition 1.2.3** (Deterministic Communication Complexity). The *deterministic communication complexity* of a problem  $\mathcal{P}$  is defined as  $D(\mathcal{P}) := \min_{\Pi: \Pi \text{ deterministically solves } \mathcal{P}} \text{CC}(\Pi)$ .

Now consider the setting where the players Alice and Bob can design their messages in the communication protocol based on the outcomes of random coin tosses. In a *private coin* protocol, each player can only see the outcomes of their own coin tosses. Formally, Alice and Bob have private random strings  $R_A$  and  $R_B$  respectively in addition to and independent of their inputs. A message that a player sends is a function of their input and their random string (and the messages in the earlier rounds of the protocol). In a *public coin* protocol, the players have access to a sufficiently long *shared* random string  $R$  that they can use to construct their messages. By *randomized* communication protocols, we refer to this stronger public coin version. In this case, the output of the protocol  $\Pi$  is a random variable denoted by  $\Pi(a, b, R)$ . We say that a randomized protocol  $\Pi$  with public random string generated from distribution  $\mathcal{D}_\Pi$  “solves a problem  $\mathcal{P}$  with error  $\delta$ ” if

$$\forall (a, b) \in A \times B : \Pr_{R \sim \mathcal{D}_\Pi} [(a, b, \Pi(a, b, R)) \notin \mathcal{P}] \leq \delta.$$

We now define the randomized communication complexity of a problem.

**Definition 1.2.4** (Randomized Communication Complexity). The  $\delta$ -error randomized communication complexity of a problem  $\mathcal{P}$  is defined as  $R_\delta(\mathcal{P}) := \min_{\Pi: \Pi \text{ solves } \mathcal{P} \text{ with error } \delta} \text{CC}(\Pi)$ .

For proving our streaming lower bounds, we are mostly interested in the  $1/3$ -error randomized communication complexity of a problem, and hence we simply define  $R(\mathcal{P}) := R_{1/3}(\mathcal{P})$ .

For proving one-pass streaming lower bounds, we shall focus on the *one-way* communication complexity of a problem, which refers to its communication complexity restricted to single-round protocols. The one-way deterministic and the one-way  $\delta$ -error randomized communication complexities of a problem  $\mathcal{P}$  are denoted by  $D^\rightarrow(\mathcal{P})$  and  $R_\delta^\rightarrow(\mathcal{P})$  respectively. The corresponding  $r$ -round complexities are denoted by  $D^r(\mathcal{P})$  and  $R_\delta^r(\mathcal{P})$ .

Let us now describe the general framework that establishes a streaming lower bound

using communication complexity. Suppose that we want to show a space lower bound for a streaming problem  $\mathcal{S}$ . Let  $\mathcal{A}$  be a  $p$ -pass streaming algorithm that solves  $\mathcal{S}$ . We consider a two-party communication problem  $\mathcal{P}$  such that the players Alice and Bob can simulate  $\mathcal{A}$  to solve  $\mathcal{P}$ : Alice feeds some stream tokens to  $\mathcal{A}$  based on her input and then sends the resulting memory state of  $\mathcal{A}$  to Bob, who continues the input stream to  $\mathcal{A}$  by feeding it tokens based on his input. If  $p > 1$ , he sends Alice the updated memory state of  $\mathcal{A}$ , and then they similarly emulate the next  $p-1$  passes of  $\mathcal{A}$ . At the end of the  $p$ th pass, Bob looks at the output of  $\mathcal{A}$  to announce a corresponding output for the communication problem  $\mathcal{P}$ , which is known to be correct if  $\mathcal{A}$  succeeds. Then, the total number of bits communicated is  $(2p-1)$  times the space  $S(\mathcal{A})$  used by  $\mathcal{A}$ . Thus, if  $\mathcal{A}$  is a randomized algorithm and succeeds with probability at least  $2/3$ , it must be that  $(2p-1) \cdot S(\mathcal{A}) \geq R^{2p-1}(\mathcal{P})$ , which implies that  $S(\mathcal{A}) \geq R^{2p-1}(\mathcal{P})/(2p-1)$ . Observe that for one-pass algorithms, we have that  $S(\mathcal{A}) \geq R^1(\mathcal{P})$ . Thus, the one-way communication complexity is useful in showing one-pass streaming lower bounds. Also, if we do not know  $R^{2p-1}(\mathcal{P})$ , but know  $R(\mathcal{P})$ , then we can weakly lower bound the former by the latter and obtain that  $S(\mathcal{A}) \geq R(\mathcal{P})/(2p-1)$ . The same framework gives analogous relations between deterministic-streaming space complexity and deterministic communication complexity.

### Section 1.3

## Notations, Terminology, and Basic Tools

**Notations and Terminology.** We fix some notations and terminology that we will be using throughout the thesis. In considering a graph coloring problem, the input graph will usually be called  $G = (V, E)$ , where  $V := V(G)$  is the set of vertices and  $E := E(G)$  is the set of edges of  $G$  (unless stated otherwise). We usually denote the number of vertices  $|V(G)|$  by  $n$  and the number of edges  $|E(G)|$  by  $m$ . When considering insert-delete graph streams, however, we usually denote the stream length by  $m$  (which is same as the number of edges

for insert-only graph streams). The notation “ $\log x$ ” stands for  $\log_2 x$ . For an integer  $k$ , we denote the set  $\{1, 2, \dots, k\}$  by  $[k]$ . We use the  $\tilde{O}()$  notation to hide factors polylogarithmic in the input size. We say that an event holds *with high probability* (w.h.p.) if the probability is at least  $1 - 1/\text{poly}(n)$ .

By *semi-streaming space*, we mean a space bound of  $O(n \cdot \text{polylog}(n))$  bits. *Semi-streaming model* refers to the streaming model where the space is restricted to  $O(n \cdot \text{polylog}(n))$  bits.

**Basic Tools.** We shall use the following form of the Chernoff bound to prove high probability statements throughout the thesis.

**Fact 1.3.1.** *Let  $X$  be a sum of mutually independent indicator random variables. Let  $\mu$  and  $\delta$  be real numbers such that  $\mathbb{E}X \leq \mu$  and  $0 \leq \delta \leq 1$ . Then,  $\Pr[X \geq (1 + \delta)\mu] \leq \exp(-\mu\delta^2/3)$ .*  $\square$

**Communication Complexity Problems for Proving Lower Bounds.** Space lower bounds for data streaming algorithms are most often proven via reductions from standard problems in communication complexity. We recall two such problems, each involving two players, Alice and Bob. In the  $\text{INDEX}_N$  problem, Alice holds a vector  $\mathbf{x} \in \{0, 1\}^N$  and Bob holds an index  $k \in [N]$ : the goal is for Alice to send Bob a message allowing him to output  $x_k$ . In the  $\text{DISJ}_N$  problem, Alice holds  $\mathbf{x} \in \{0, 1\}^N$  and Bob holds  $\mathbf{y} \in \{0, 1\}^N$ : the goal is for them to communicate interactively, following which they must decide whether  $\mathbf{x}$  and  $\mathbf{y}$  are disjoint, when considered as subsets of  $[N]$ , i.e., they must output  $\neg \bigvee_{i=1}^N x_i \wedge y_i$ . In the special case  $\text{DISJ}_{N,s}$ , it is promised that the cardinalities  $|\mathbf{x}| = |\mathbf{y}| = s$ . In each case, the communication protocol may be randomized, erring with probability at most  $\delta$ . We shall use the following well-known lower bounds.

**Fact 1.3.2** ([3]). *The one-way randomized complexity  $R^\rightarrow(\text{INDEX}_N) = \Omega(N)$ .*

**Fact 1.3.3** ([153]). *The general randomized complexity  $R(\text{DISJ}_{N,N/3}) = \Omega(N)$ .*

---

## Chapter 2

---

# Classical Graph Streaming

A data stream is a sequence  $\langle a_1 \dots, a_m \rangle$  where each  $a_i$  comes from a universe  $U$ . A data streaming algorithm reads such a stream and maintains a summary using space sublinear in the input size, i.e.,  $o(m)$ , so as to compute some given function of the stream. An algorithm can make one or a few passes over the stream to compute the summary. At the end of the last pass, it does some post-processing on the summary to return an output. The time taken for post-processing or for processing an update are usually not the focus of optimization for this model; it just aims to optimize the space usage and the number of passes.

In this chapter, we study graph streaming, i.e., we focus on graph problems in this model. Suppose that the input graph  $G = (V, E)$  is on  $n$  nodes. To define a graph stream most generally, each stream token  $a_i$  is of the form  $((u, v), c)$  which denotes that  $c$  edges are inserted (resp. deleted) between vertices  $u, v \in V$  if  $c > 0$  (resp.  $c < 0$ ). The simplest case is when  $c$  is always 1, i.e., the graph is simple and the input stream is insert-only. In this case, the stream length  $m = O(n^2)$ . We mostly consider this case while trying to establish lower bounds, since such lower bounds would be strong. When  $c = \pm 1$ , i.e., the graph is simple but both edge insertions and deletions are allowed, the stream length  $m$  can be potentially much larger than  $\Theta(n^2)$ . In this case, rather than  $o(m)$ , we naturally aim for  $o(n^2)$  space (since we can always store the entire graph in  $\Theta(n^2)$  space in the worst case).

We call such streams as *dynamic* or *insert-delete* graph streams. For some problems, we allow multigraphs, i.e., a vertex pair can share multiple parallel edges. Here,  $c$  can be any integer. Some of our algorithms can even handle such graphs. However, we do not allow the rather impractical case of “negative edges” caused by a stream trying to delete an edge that does not exist in the current graph.

It often helps to look at a streaming graph as updates made to its characteristic vector  $\mathbf{e}$  of length  $\binom{n}{2}$  (for undirected graphs) or  $n(n-1)$  (for directed graphs), where for some canonical indexing of the vertex pairs,  $e_i$  represents the number of edges shared between the  $i$ th vertex pair. In the case that the graph is simple, the vector is a binary string. Since we do not allow deletion of an edge before it is inserted, the entries of the characteristic vector are always non-negative (even for the most general multigraph case) and hence, we are considering the so called *strict turnstile* data streaming model.

Ideally, we would like to solve a problem using space  $\text{polylog}(n)$ , but for most graph problems, this is too much to ask for. Instead, a space bound that we usually aim for is  $\tilde{O}(n) = O(n \cdot \text{polylog}(n))$ . This makes many graph problems tractable as it allows us to store non-zero (though small) information about each node. At the same time, the bound is sublinear in  $\Theta(n^2)$ , the worst-case size of the graph. As noted in Section 1.3, an algorithm using this much space is called a semi-streaming algorithm and the graph streaming model restricted to this much space is called the semi-streaming model [81]. For the last two decades, the semi-streaming model has been the most popular setting for the study of graph streaming. That said, space complexity of  $O(n^{1+\alpha})$  for  $\alpha < 1$  is also an interesting regime to study.

The order of the input stream often turns out to be crucial. In Section 2.1, we consider both cases: (i) adversarial order, where the order of the stream is determined by an adversary before the algorithm reads it, and (ii) random order, where the stream order is taken uniformly at random from the set of  $m!$  possible permutations of the stream. We exhibit

problems whose complexity drastically changes with the stream order as well as problems whose complexity does not undergo any significant change. Note that for most problems (even for those on random-order streams), we assume that the underlying input is a fixed adversarially chosen graph; the order of presentation of its edges can then be either adversarial or random. We, however, consider a couple of cases where the graph itself is drawn from a random distribution.

In Section 2.1, we present results on a collection of problems on directed graph streams. A common theme for these problems is that they deal with vertex orderings. In Section 2.2, we study an undirected graph problem where vertex orderings turn out to be important. This is the graph coloring problem with respect to a graph parameter called degeneracy (see Section 2.2 for definition), for which we devise efficient algorithms and prove space lower bounds.

## Section 2.1

### Directed Graph Problems

While there has been a large body of work on undirected graphs in the data stream model [137], the complexity of processing directed graphs (digraphs) in this model was relatively unexplored prior to our results. The handful of exceptions include multipass algorithms emulating random walks in directed graphs [107, 156], establishing prohibitive space lower bounds on finding sinks [103] and answering reachability queries [81], and ruling out semi-streaming constant-pass algorithms for directed reachability [95]. This is rather unfortunate given that many of the massive graphs often mentioned in the context of motivating work on graph streaming are directed, e.g., hyperlinks, citations, and social media “follows” all correspond to directed edges.

In this section, we consider the complexity of a variety of fundamental problems related

to vertex ordering in directed graphs. For example, one basic problem that motivated<sup>1</sup> much of the work in this section is as follows: given a stream consisting of edges of an acyclic graph in an arbitrary order, how much memory is required to return a topological ordering of the graph? In the offline setting, this can be computed in  $O(m + n)$  time using Kahn’s algorithm [110] or via depth-first trees [162] but nothing was known in the data stream setting.

As another example, consider the related minimum feedback arc set problem, i.e., estimating the minimum number of edges (arcs) that need to be removed to make the resulting graph acyclic. This problem is NP-hard and to the best of our knowledge, the best known approximation factor is  $O(\log n \log \log n)$  for arbitrary graphs [78], although a PTAS is known in the case of tournaments [118]. Again, nothing was known in the data stream model. In contrast, the analogous problem for undirected graphs is well understood in streaming. The number of edges required to make an undirected graph acyclic is  $m - n + c$  where  $c$  is the number of connected components. The number of connected components can be computed in  $O(n \log n)$  space by constructing a spanning forest [5, 81].

### 2.1.1. Our Results

We describe our results for digraph problems. A summary is given in Table 2.1.

**Arbitrary Graphs.** In Section 2.1.4 we present a number of negative results for the case when the input digraph can be arbitrary. In particular, we show that there is no one-pass sublinear-space algorithm for such fundamental digraph problems as testing whether an input digraph is acyclic, topologically sorting it if it is, or finding its feedback arc set if it is not. These results set the stage for our later focus on specific families of graphs, where we can do much more, algorithmically.

For our lower bounds, we consider both arbitrary and random stream orderings. In Sec-

---

<sup>1</sup>The problem was explicitly raised in an open problems session at the Shonan Workshop “Processing Big Data Streams” (June 5-8, 2017) and generated considerable discussion.

| Problem                           | Passes      | Space Bound                      | Notes                         |
|-----------------------------------|-------------|----------------------------------|-------------------------------|
| ACYC                              | 1           | $\Theta(n^2)$                    |                               |
| ACYC                              | $p$         | $n^{1+\Omega(1/p)}/p^{O(1)}$     |                               |
| mult. approx. FAS-SIZE            | 1           | $\Theta(n^2)$                    |                               |
| mult. approx. FAS-SIZE            | $p$         | $n^{1+\Omega(1/p)}/p^{O(1)}$     |                               |
| TOPO-SORT                         | 1           | $\Theta(n^2)$                    |                               |
| TOPO-SORT                         | $p$         | $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ |                               |
| mult. approx. FAS                 | 1           | $\Theta(n^2)$                    |                               |
| mult. approx. FAS                 | $p$         | $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ |                               |
| STCONN-DAG (RO)                   | $p$         | $n^{1+\Omega(1/p)}/p^{O(1)}$     | error prob. $1/p^{\Omega(p)}$ |
| ACYC (RO)                         | $p$         | $n^{1+\Omega(1/p)}/p^{O(1)}$     | error prob. $1/p^{\Omega(p)}$ |
| mult. approx. FAS-SIZE (RO)       | $p$         | $n^{1+\Omega(1/p)}/p^{O(1)}$     | error prob. $1/p^{\Omega(p)}$ |
| TOPO-SORT (RO)                    | $p$         | $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ | error prob. $1/p^{\Omega(p)}$ |
| mult. approx. FAS (RO)            | $p$         | $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ | error prob. $1/p^{\Omega(p)}$ |
| $(1+\varepsilon)$ -approx. FAS-T  | 1           | $\tilde{O}(\varepsilon^{-2}n)$   | exp. time post-processing     |
| 3-approx. FAS-T                   | $p$         | $\tilde{O}(n^{1+1/p})$           |                               |
| ACYC-T                            | 1           | $\tilde{O}(n)$                   |                               |
| ACYC-T                            | $p$         | $\Omega(n/p)$                    |                               |
| SINK-FIND-T                       | $2p-1$      | $\tilde{O}(n^{1/p})$             |                               |
| SINK-FIND-T                       | $p$         | $\Omega(n^{1/p}/p^2)$            |                               |
| SINK-FIND-T (RO)                  | 1           | $\tilde{O}(1)$                   |                               |
| TOPO-SORT (RO)                    | 1           | $\tilde{O}(n^{3/2})$             | random DAG + planted path     |
| TOPO-SORT                         | $O(\log n)$ | $\tilde{O}(n^{4/3})$             | random DAG + planted path     |
| $(1+\varepsilon)$ -apx. RANK-AGGR | 1           | $\tilde{O}(\varepsilon^{-2}n)$   | exp. time post-processing     |

Table 2.1: Summary of our algorithmic and space lower bound results. These problems are defined in Section 2.1.3. The input stream is adversarially ordered unless marked with (RO) which stands for “Random Order”. Besides the above results, we also give an oracle (query complexity) lower bound in Section 2.1.6.

tion 2.1.4, we concentrate on arbitrary orderings and show that checking whether the graph is acyclic, finding a topological ordering of a directed acyclic graph (DAG), or any multiplicative approximation of feedback arc set requires  $\Omega(n^2)$  space in one pass. The lower bound extends to  $n^{1+\Omega(1/p)}/p^{O(1)}$  when the number of passes is  $p \geq 1$ . In Section 2.1.4, we show that essentially the same bound holds even when the stream is *randomly* ordered. This

strengthening is one of our more technically involved results and it is based on generalizing a fundamental result by Guruswami and Onak [95] on  $s$ – $t$  connectivity in the multi-pass data stream model.

As a by-product of our generalization, we also obtain the first random-order lower bounds refuting semi-streaming space for *undirected* graph problems; these include deciding (i) whether there exists a short  $s$ – $t$  path, and (ii) whether there exists a perfect matching.

**Tournaments.** A *tournament* is a digraph that has exactly one directed edge between each pair of distinct vertices. In Section 2.1.5, we consider the problem of finding a sink in a tournament which is guaranteed to be acyclic. Obviously, this problem can be solved in a single pass using  $O(n)$  space by maintaining an “is-sink” flag for each vertex. Our results show that for arbitrary order streams this is tight. We prove that finding a sink in  $p$  passes requires  $\Omega(n^{1/p}/p^2)$  space. For upper bounds, we provide an  $O(n^{1/p} \log(3p))$ -space sink-finding algorithm that uses  $O(p)$  passes, for any  $1 \leq p \leq \log n$ . In contrast, we show that if the stream is randomly ordered, then just a single pass using only  $\text{polylog } n$  space is sufficient. This is a significant separation between the arbitrary-order and random-order data stream models.

If we assume that the input graph is a tournament, it is trivial to find a topological ordering, given that one exists, by considering the in-degrees of the vertices. Furthermore, it is known that ordering the vertices by in-degree yields a 5-approximation to feedback arc set [69]. In Section 2.1.6, we present an algorithm which computes a  $(1 + \varepsilon)$ -approximation to feedback arc set in one pass using  $\tilde{O}(\varepsilon^{-2}n)$  space. In the post-processing step, however, it estimates the number of back edges for every permutation of vertices in the graph, thus resulting in exponential post-processing time. Despite its “brute force” feel, our algorithm is essentially optimal, both in its space usage (unconditionally) and its post-processing time (in a sense we shall make precise later). We address these issues in Section 2.1.6. On the other hand, in Section 2.1.6, we show that with  $O(\log n)$  additional passes it is possible to

compute a 3-approximation to feedback arc set while using only polynomial time and  $\tilde{O}(n)$  space.

**Random Graphs.** In Section 2.1.7, we consider a natural family of random acyclic graphs (see Definition 2.1.2 below) and present two algorithms for finding a topological ordering of vertices. We show that, for this family,  $\tilde{O}(n^{4/3})$  space is sufficient to find the best ordering given  $O(\log n)$  passes. Alternatively,  $\tilde{O}(n^{3/2})$  space is sufficient given only a single pass, on the assumption that the edges in the stream are randomly ordered. These results show that for proving stronger lower bounds for topological sorting, the standard technique showing hardness for such random inputs and then applying Yao’s lemma would not work; we need more involved input distributions or more sophisticated techniques.

**Rank Aggregation.** In Section 2.1.8, we consider the problem of rank aggregation (formally defined in the next section), which is closely related to the feedback arc set problem. We present a one-pass  $\tilde{O}(\varepsilon^{-2}n)$ -space algorithm that returns  $(1 + \varepsilon)$ -approximation to the rank aggregation problem. The algorithm is very similar to our  $(1 + \varepsilon)$ -approximation of feedback arc set in tournaments and has the same drawback of using exponential post-processing time.

### 2.1.2. Previous Work

Some versions of the problems we study here have been considered previously in the query complexity model. For example, Huang et al. [104] consider the “generalized sorting problem” where  $G$  is an acyclic graph with a unique topological order. The algorithm is presented with an undirected version of this graph and may query any edge to reveal its direction. The goal is to learn the topological ordering with the minimum number of queries. Huang et al. [104] and Angelov et al. [11] also studied the average case complexity of various problems where the input graph is chosen from some known distribution. Ailon [6] studied the equivalent problem for feedback arc set in tournaments. Note that all these

query complexity results are adaptive and do not immediately give rise to small-space data stream algorithms.

Perhaps the relative lack of progress on streaming algorithms for directed graph problems stems from their being considered “implicitly hard” in the literature, a point made in the recent work of Khan and Mehta [119]. Indeed, that work and the also-recent work of Elkin [75] provide the first nontrivial streaming algorithms for computing a depth-first search tree and a shortest-paths tree (respectively) in semi-streaming space, using  $O(n/\text{polylog } n)$  passes. Notably, fairly non-trivial work was needed to barely beat the trivial bound of  $O(n)$  passes.

Some of our work here applies and extends the work of Guruswami and Onak [95], who gave the first lower bounds precluding semi-streaming space for decision problems on graphs. In particular, they showed that solving reachability in  $n$ -vertex digraphs using  $p$  passes requires  $n^{1+\Omega(1/p)}/p^{O(1)}$  space. Via simple reductions, they then showed similar lower bounds for deciding whether a given (undirected) graph has a short  $s$ – $t$  path or a perfect matching.

### 2.1.3. Problems and Preliminaries

**Vertex Ordering Problems in Digraphs.** An *ordering* of an  $n$ -vertex digraph  $G = (V, E)$  is a list consisting of its vertices. We shall view each ordering  $\sigma$  as a function  $\sigma: V \rightarrow [n]$ , with  $\sigma(v)$  being the position of  $v$  in the list. To each ordering  $\sigma$ , there corresponds a set of *back edges*  $B_G(\sigma) = \{(v, u) \in E : \sigma(u) < \sigma(v)\}$ . We say that  $\sigma$  is a *topological ordering* if  $B_G(\sigma) = \emptyset$ ; such  $\sigma$  exists iff  $G$  is acyclic. We define  $\beta_G = \min\{|B_G(\sigma)| : \sigma \text{ is an ordering of } G\}$ , i.e., the size of a minimum feedback arc set for  $G$ .

We now define the many interrelated digraph problems studied in this work. In each of these problems, the input is a digraph  $G$ , presented as a stream of its edges. The ordering of the edges is adversarial unless specified otherwise.

ACYC: Decide whether or not  $G$  is acyclic.

TOPO-SORT: Under the promise that  $G$  is acyclic, output a topological ordering of its vertices.

STCONN-DAG: Under the promise that  $G$  is acyclic, decide whether it has an  $s$ -to- $t$  path, these being two prespecified vertices.

SINK-FIND: Under the promise that  $G$  is acyclic, output a sink of  $G$ .

FAS-SIZE ( $\alpha$ -approximation): Output an integer  $\hat{\beta} \in [\beta_G, \alpha\beta_G]$ .

FAS ( $\alpha$ -approximation): Output an ordering  $\sigma$  such that  $|B_G(\sigma)| \leq \alpha\beta_G$ .

FAS-T: Solve FAS under the promise that  $G$  is a tournament. In a similar vein, we define the promise problems ACYC-T, TOPO-SORT-T, SINK-FIND-T, FAS-SIZE-T.

For randomized solutions to these problems we shall require that the error probability be at most  $1/3$ .

We remark that the most common definition of the minimum feedback arc set problem in the literature on optimization is to identify a small set of edges whose removal makes the graph acyclic, so FAS-SIZE is closer in spirit to this problem than FAS. As we shall see, our algorithms will apply to both variants of the problem. On the other hand, lower bounds sometimes require different proofs for the two variants. Since  $\beta_G = 0$  iff  $G$  is acyclic, we have the following basic observation.

**Observation 2.1.1.** *Producing a multiplicative approximation for any of FAS, FAS-T, FAS-SIZE, and FAS-SIZE-T entails solving (respectively) TOPO-SORT, TOPO-SORT-T, ACYC, and ACYC-T.*

For an ordering  $\pi$  of a vertex set  $V$ , define  $E^\pi = \{(u, v) \in V^2 : \pi(u) < \pi(v)\}$ . Define  $\text{Tou}(\pi) = (V, E^\pi)$  to be the unique acyclic tournament on  $V$  consistent with  $\pi$ .

As mentioned above, we will also consider vertex ordering problems on random graphs from a natural distribution. This distribution, which we shall call a “planted path distri-

bution,” was considered by Huang et al. [104] for average case analysis in their work on generalized sorting.

**Definition 2.1.2** (Planted Path Distribution). Let  $\text{PlantDAG}_{n,q}$  be the distribution on digraphs on  $[n]$  defined as follows. Pick a permutation  $\pi$  of  $[n]$  uniformly at random. Retain each edge  $(u, v)$  in  $\text{Tou}(\pi)$  with probability 1 if  $\pi(v) = \pi(u) + 1$ , and with probability  $q$ , independently, otherwise.

**Rank Aggregation.** The feedback arc set problem in tournaments is closely related to the problem of *rank aggregation* (RANK-AGGR). Given  $k$  total orderings  $\sigma_1, \dots, \sigma_k$  of  $n$  objects we want to find an ordering that best describes the “preferences” expressed in the input. Formally, we want to find an ordering that minimizes  $\text{cost}(\pi) := \sum_{i=1}^k d(\pi, \sigma_i)$ , where the distance  $d(\pi, \sigma)$  between two orderings is the number of pairs of objects ranked differently by them. That is,

$$d(\pi, \sigma) := \sum_{a, b \in [n]} \mathbb{1}\{\pi(a) < \pi(b), \sigma(b) < \sigma(a)\},$$

where the notation  $\mathbb{1}\{\phi\}$  denotes a 0/1-valued indicator for the condition  $\phi$ .

In the streaming model, the input to RANK-AGGR can be given either as a concatenation of  $k$  orderings, leading to a stream of length  $kn$ , or as a sequence of triples  $(a, b, i)$  conveying that  $\sigma_i(a) < \sigma_i(b)$ , leading to a stream of length  $k\binom{n}{2}$ . Since we want the length of the stream to be polynomial in  $n$ , we assume  $k = n^{O(1)}$ .

#### 2.1.4. General Digraphs and the Hardness of Some Basic Problems

In this section, our focus is bad news. In particular, we show that there is no one-pass sublinear-space algorithm for the rather basic problem of testing whether an input digraph is acyclic, nor for topologically sorting it if it is. These results set the stage for our later focus on *tournament* graphs, which do allow interesting algorithms for these problems.

**Arbitrary Order Lower Bounds.** To begin, note that the complexity of TOPO-SORT-T is easily understood: maintaining in-degrees of all vertices and then sorting by in-degree provides a one-pass  $O(n \log n)$ -space solution. However, the problem becomes maximally hard without the promise of a tournament.

**Theorem 2.1.3.** *Solving TOPO-SORT in one pass requires  $\Omega(n^2)$  space.*

*Proof.* We reduce from INDEX<sub>N</sub>, where  $N = p^2$  for a positive integer  $p$ . Using a canonical bijection from  $[p]^2$  to  $[N]$ , we rewrite Alice's input vector as a matrix  $\mathbf{x} = (\mathbf{x}_{ij})_{i,j \in [p]}$  and Bob's input index as  $(y, z) \in [p]^2$ . Our reduction creates a graph  $G = (V, E)$  on  $n = 4p$  vertices: the vertex set  $V = L^0 \uplus R^0 \uplus L^1 \uplus R^1$ , where each  $|L^b| = |R^b| = p$ . These vertices are labeled, with  $\ell_i^0$  being the  $i$ th vertex in  $L^0$  (and similarly for  $r_i^0, \ell_i^1, r_i^1$ ).

Based on their inputs, Alice and Bob create streams of edges by listing the following sets:

$$E_{\mathbf{x}} = \{(\ell_i^b, r_j^b) : b \in \{0, 1\}, i, j \in [p], x_{ij} = b\}, \quad E_{yz} = \{(r_z^0, \ell_y^1), (r_z^1, \ell_y^0)\}.$$

The combined stream defines the graph  $G$ , where  $E = E_{\mathbf{x}} \cup E_{yz}$ .

We claim that  $G$  is acyclic. In the digraph  $(V, E_{\mathbf{x}})$ , every vertex is either a source or a sink. So the only vertices that could lie on a cycle in  $G$  are  $\ell_y^0, r_z^0, \ell_y^1$ , and  $r_z^1$ . Either  $(\ell_y^0, r_z^0) \notin E$  or  $(\ell_y^1, r_z^1) \notin E$ , so there is in fact no cycle even among these four vertices.

Let  $\sigma$  be a topological ordering of  $G$ . If  $x_{yz} = 0$ , then we must, in particular, have  $\sigma(\ell_y^0) < \sigma(\ell_y^1)$ , else we must have  $\sigma(\ell_y^1) < \sigma(\ell_y^0)$ . Thus, by simulating a one-pass algorithm  $\mathcal{A}$  on Alice's stream followed by Bob's stream, consulting the ordering  $\sigma$  produced by  $\mathcal{A}$  and outputting 0 iff  $\sigma(\ell_y^0) < \sigma(\ell_y^1)$ , the players can solve INDEX<sub>N</sub>. It follows by Fact 1.3.2 that the space used by  $\mathcal{A}$  must be at least  $R^+(\text{INDEX}_N) = \Omega(N) = \Omega(p^2) = \Omega(n^2)$ .  $\square$

For our next two results, we use reductions from STCONN-DAG. It is a simple exercise to show that a one-pass streaming algorithm for STCONN-DAG requires  $\Omega(n^2)$  space.

Guruswami and Onak [95] showed that a  $p$ -pass algorithm requires  $n^{1+\Omega(1/p)}/p^{O(1)}$  space.<sup>2</sup>

**Proposition 2.1.4.** *Solving ACYC requires  $\Omega(n^2)$  space in one pass and  $n^{1+\Omega(1/p)}/p^{O(1)}$  space in  $p$  passes.*

*Proof.* Given a DAG  $G$  and specific vertices  $s, t$ , let  $G'$  be obtained by adding edge  $(t, s)$  to  $G$ . Then  $G'$  is acyclic iff  $G$  has no  $s$ -to- $t$  path. By the discussion above, the lower bounds on ACYC follow.  $\square$

**Corollary 2.1.5.** *Solving TOPO-SORT in  $p$  passes requires  $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$  space.*

*Proof.* Given a  $p$ -pass  $S$ -space algorithm  $A$  for TOPO-SORT, we can obtain a  $(p+1)$ -pass  $(S + O(n \log n))$ -space algorithm for ACYC as follows. Run algorithm  $A$ , store the ordering it outputs, and in another pass, check if the ordering induces any back-edge. If it does, we output NO, and otherwise, we output YES. In case of any runtime error, we return NO. For correctness, observe that if the input graph  $G$  is acyclic, then  $A$  returns a valid topological ordering w.h.p.. Hence, the final pass doesn't detect any back-edge, and we correctly output YES. In case  $G$  is not acyclic, the promise that the input graph for TOPO-SORT would be a DAG is violated, and hence,  $A$  either raises an error or returns some ordering that must induce a back-edge since  $G$  doesn't have a topological ordering. Thus, we correctly return NO in this case. Finally, Proposition 2.1.4 implies that  $S + O(n \log n) \geq n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ , i.e.,  $S \geq n^{1+\Omega(1/p)}/(p+1)^{O(1)}$ .  $\square$

**Corollary 2.1.6.** *A multiplicative approximation algorithm for either FAS-SIZE or FAS requires  $\Omega(n^2)$  space in one pass. In  $p$  passes, such approximations require  $n^{1+\Omega(1/p)}/p^{O(1)}$  space and  $n^{1+\Omega(1/p)}/(p+1)^{O(1)}$  space respectively.*

*Proof.* This is immediate from Observation 2.1.1, Theorem 2.1.3, Proposition 2.1.4, and Corollary 2.1.5.  $\square$

---

<sup>2</sup>Although their paper states the lower bound for  $s$ - $t$  connectivity in general digraphs, their proof in fact shows the stronger result that the bound holds even when restricted to DAGs.

*Remark.* Subsequent works have improved the multipass lower bound for adversarial order STCONN-DAG, which in turn, improves the above lower bounds for TOPO-SORT, ACYC, FAS, and FAS-SIZE. See Section 2.1.9 for a detailed discussion.

**Random Order Lower Bounds.** We consider the STCONN-DAG, ACYC, and FAS problems in a uniformly randomly ordered digraph stream. Recall that for adversarially ordered streams, these problems require about  $n^{1+\Omega(1/p)}$  space in  $p$  passes. The hardness ultimately stems from a similar lower bound for the SHORTPATH-DAG problem. In this latter problem, the input is an  $n$ -vertex DAG with two designated vertices  $v_s$  and  $v_t$ , such that either (a) there exists a path of length at most  $2p + 2$  from  $v_s$  to  $v_t$  or (b)  $v_t$  is unreachable from  $v_s$ . The goal is to determine which of these is the case.

Our goal in this section is to show that the same lower bound continues to hold under random ordering, provided we insist on a sufficiently small error probability, about  $1/p^{\Omega(p)}$ . We prove this for SHORTPATH-DAG. As this is a special case of STCONN-DAG, a lower bound for SHORTPATH-DAG carries over to STCONN-DAG. Further, by the reductions in Proposition 2.1.4 and Corollaries 2.1.5 and 2.1.6, the lower bounds also carry over to ACYC, TOPO-SORT, and FAS. We also show a barrier result arguing that this restriction to low error is necessary: for the SHORTPATH-DAG problem, if an error probability of at least  $2/p!$  is allowed, then  $\tilde{O}(n)$  space is achievable in  $p$  passes.

Our proof uses the machinery of the Guruswami–Onak lower bound for SHORTPATH-DAG under an adversarial stream ordering [95]. As in their work, we derive our space lower bound from a communication lower bound for *set chasing intersection* (henceforth, SCI). However, unlike them, we need to prove a “robust” lower bound for SCI, in the sense of Chakrabarti, Cormode, and McGregor [56], as explained below. To define SCI, we first set up a special family of multilayer pointer jumping problems, described next.

Picture a layered digraph  $G^*$  with  $2k+1$  layers of vertices, each layer having  $m$  vertices, laid out in a rectangular grid with each column being one layer. From left to right, the layers

are numbered  $-k, -k+1, \dots, k$ . Layer 0 is called the *mid-layer*. The only possible edges of  $G^*$  are from layer  $\ell$  to layer  $\ell-1$ , or from layer  $-\ell$  to layer  $-\ell+1$ , for  $\ell \in [k]$  (i.e., edges travel from the left and right ends of the rectangular grid towards the mid-layer). We designate the first vertex in layer  $-k$  as  $v_s$  and the first vertex in layer  $k$  as  $v_t$ .

Each vertex not in the mid-layer has exactly  $t$  outgoing edges, numbered 1st through  $t$ th, possibly with repetition (i.e.,  $G^*$  is a multigraph). Think of these edges as *pointers*. An input to one of our communication problems (to be defined soon) specifies the destinations of these pointers. Thus, an input consists of  $2mkt$  *tokens*, where each token is an integer in  $[m]$  specifying which of the  $m$  possibilities a certain pointer takes. The pointers emanating from layer  $\ell$  of vertices constitute the  $\ell$ th layer of pointers. Our communication games will involve  $2k$  players named  $P_{-k}, \dots, P_{-1}, P_1, \dots, P_k$ . We say that  $P_\ell$  is the *natural owner* of the portion of the input specifying the  $\ell$ th layer of pointers.

In the  $\text{SCI}_{m,k,t}$  problem, the goal is to determine whether or not there exists a mid-layer vertex reachable from  $v_s$  as well as  $v_t$ . Consider the communication game where each pointer is known to its natural owner and the players must communicate in  $k-1$  rounds, where in each round they broadcast messages in the fixed order  $P_{-1}, \dots, P_{-k}, P_1, \dots, P_k$ . Guruswami and Onak [95] showed that this problem requires a total communication cost of  $\Omega(m^{1+1/(2k)}/k^{16} \log^{3/2} m)$  in the parameter regime  $t^{2k} \ll m$ . This almost immediately implies a similar lower bound for  $\text{SHORTPATH-DAG}$ —simply reverse the directions of the pointers in positive-numbered layers—which then translates into a data streaming lower bound along standard lines.

The key twist in *our* version of the SCI problem is that each pointer is allocated to one of the  $2k$  players *uniformly at random*: thus, most pointers are not allocated to their natural owners. The players have to determine the output to SCI communicating exactly in the same pattern as before, up to a small error probability taken over the protocol’s internal randomness as well as the random allocation. This setup potentially makes the problem

easier because there is a good chance that the players will be able to “jump two pointers” within a single round. Our main technical result is to show that a lower bound of the form  $m^{1+\Omega(1/k)}$  holds despite this. In the terminology of Chakrabarti et al. [56], who lower-bounded a number of communication problems under such random-allocation setups, this is a *robust* communication lower bound.

**Theorem 2.1.7.** *Suppose that  $t^{2k} = o(m / \text{polylog}(m))$  and that protocol  $\Pi$  solves  $\text{SCI}_{m,k,t}$  with error  $\varepsilon < (2k)^{-2k-2}$  when the input is randomly allocated amongst the  $2k$  players, as described above. Then,  $\Pi$  communicates  $\Omega(m^{1+1/(2k)} / k^{16} \log^{3/2} m)$  bits.*

To prove this result, we consider a problem we call  $\text{MPJ-MEET}_{m,k,t}$ , defined next (Guruswami and Onak called this problem  $\text{OR} \circ \text{LPCE}$ ). Consider an input  $G^*$  to  $\text{SCI}_{m,k,t}$  and fix an  $i \in [t]$ . If we retain only the  $i$ th pointer emanating from each vertex, for each layer  $\ell$ , the  $\ell$ th layer of pointers defines a function  $f_{\ell,i}: [n] \rightarrow [n]$ . Let  $x_i$  (respectively,  $y_i$ ) denote the index of the unique mid-layered vertex reached from  $v_s$  (respectively,  $v_t$ ) by following the retained pointers. Formally,

$$x_i = f_{-1,i}(f_{-2,i}(\cdots f_{-k,i}(1) \cdots)), \quad y_i = f_{1,i}(f_{2,i}(\cdots f_{k,i}(1) \cdots)).$$

Define a function to be  $r$ -thin if every element in its range has at most  $r$  distinct pre-images. The instance  $G^*$  is said to *meet at  $i$*  if  $x_i = y_i$  and is said to be  $r$ -thin at  $i$  if each function  $f_{\ell,i}$  is  $r$ -thin. The desired output of  $\text{MPJ-MEET}$  is

$$\text{MPJ-MEET}(G^*) = \bigvee_{i=1}^t \mathbb{1}\{G^* \text{ meets at } i\} \vee \mathbb{1}\{G^* \text{ is not } (C \log m)\text{-thin at } i\},$$

for an appropriate universal constant  $C$ . The corresponding communication game allocates each pointer to its natural owner and asks them to determine the output using the same communication pattern as for  $\text{SCI}$ . Here is the key result about this problem.

**Lemma 2.1.8** (Lemma 7 of Guruswami–Onak [95]). *The  $(k - 1)$ -round constant-error communication complexity  $R^{k-1}(\text{MPJ-MEET}_{m,k,t}) = \Omega(tm/(k^{16} \log m)) - O(kt^2)$ .  $\square$*

Using this, we prove our main technical result.

*Proof of Theorem 2.1.7.* Based on the  $\varepsilon$ -error protocol  $\Pi$  for  $\text{SCI}_{m,k,t}$ , we design a protocol  $\mathcal{Q}$  for  $\text{MPJ-MEET}_{m,k,t}$  as follows. Let  $G^*$  be an instance of  $\text{MPJ-MEET}$  allocated to players as described above. The players first check whether, for some  $i$ ,  $G^*$  fails to be  $r$ -thin at  $i$ , for  $r := C \log m$ : this check can be performed in the first round of communication with each player communicating a single bit. If the check passes, the protocol ends with output 1. From now on, we assume that  $G^*$  is indeed  $r$ -thin at each  $i \in [t]$ .

Using public randomness, the players randomly renumber the vertices in each layer of  $G^*$ , creating an instance  $G'$  of  $\text{SCI}$ .<sup>3</sup> The players then choose  $\rho$ , a random allocation of pointers as in the  $\text{SCI}$  problem. They would like to simulate  $\Pi$  on  $G'$ , as allocated by  $\rho$ , but of course they can't do so without additional communication. Instead, using further public randomness, for each pointer that  $\rho$  allocates to someone besides its natural owner, the players reset that pointer to a uniformly random (and independent) value in  $[m]$ . We refer to such a pointer as *damaged*. Since there are  $2k$  players, each pointer is damaged with probability  $1 - 1/(2k)$ . Let  $G''$  denote the resulting random instance of  $\text{SCI}$ . The players then simulate  $\Pi$  on  $G''$  as allocated by  $\rho$ .

It remains to analyze the correctness properties of  $\mathcal{Q}$ . Suppose that  $G^*$  is a 1-instance of  $\text{MPJ-MEET}$ . Then there exists  $i \in [t]$  such that  $G^*$  meets at  $i$ . By considering the unique maximal paths out of  $v_s$  and  $v_t$  following only the  $i$ th pointers at each vertex, we see that  $G^*$  is also a 1-instance of  $\text{SCI}$ . Since the vertex renumbering preserves connectivity,  $G'$  is also a 1-instance of  $\text{SCI}$ . With probability  $(2k)^{-2k}$ , none of the  $2k$  pointers on these renumbered paths is damaged; when this event occurs,  $G''$  is also a 1-instance of  $\text{SCI}$ . Therefore,  $\mathcal{Q}$

<sup>3</sup>This step is exactly as in Guruswami–Onak [95]. Formally, each function  $f_{\ell,i}$  is replaced by a corresponding function of form  $\pi_{\ell,i} \circ f_{\ell,i} \circ \pi_{\ell+1,i}^{-1}$  (for  $\ell > 0$ ), for random permutations  $\pi_{\ell,i}: [m] \rightarrow [m]$ . To keep things concise, we omit the full details here.

outputs 1 with probability at least  $(2k)^{-2k}(1 - \text{err}(\Pi)) \geq (2k)^{-2k}(1 - \varepsilon)$ .

Next, suppose that  $G^*$  is a 0-instance of MPJ-MEET. It could be that  $G^*$  is a 1-instance of SCI. However, as Guruswami and Onak show,<sup>4</sup> the random vertex renumbering ensures that  $\Pr[\text{SCI}(G') = 1] < o(1)$ . For the rest of the argument, assume that  $\text{SCI}(G') = 0$ . In order to have  $\text{SCI}(G'') = 1$ , there must exist a mid-layer vertex  $x$  such that

$$f_{1,i_1}(f_{2,i_2}(\cdots f_{k,i_k}(1)\cdots)) = x = f_{-1,j_1}(f_{-2,j_2}(\cdots f_{-k,j_k}(1)\cdots)) \quad (2.1)$$

for some choice of pointer numbers  $i_1, \dots, i_k, j_1, \dots, j_k \in [t]$ . We consider three cases.

- *Case 1: None of the pointers in the above list is damaged.* In this case, eq. (2.1) cannot hold, because  $\text{SCI}(G') = 0$ .
- *Case 2: The layer-1 pointer in the above list is damaged.* Condition on a particular realization of pointers in negative-numbered layers and let  $x$  denote the mid-layered vertex reached from  $v_s$  by following pointers numbered  $j_k, \dots, j_1$ , as in eq. (2.1). The probability that the damaged pointer at layer 1 points to  $x$  is  $1/m$ . Since this holds for each conditioning, the probability that  $\text{SCI}(G'') = 1$  is also  $1/m$ .
- *Case 3: The layer- $\ell$  pointer is damaged, but pointers in layers  $1, \dots, \ell - 1$  are not, where  $\ell \geq 2$ .* Again, condition on a particular realization of pointers in negative-numbered layers and let  $x$  be as above. Since the functions  $f$  in eq. (2.1) are all  $r$ -thin, the number of vertices in layer  $\ell - 1$  that can reach  $x$  using only undamaged pointers is at most  $r^{\ell-1} \leq r^{k-1}$ . The probability that the damaged pointer at layer  $\ell$  points to one of these vertices is at most  $r^{k-1}/m$ .

Combining the cases, the probability that eq. (2.1) holds for a particular choice of pointer numbers  $i_1, \dots, i_k, j_1, \dots, j_k \in [t]$  is at most  $r^{k-1}/m$ . Taking a union bound over

---

<sup>4</sup>See the final paragraph of the proof of Lemma 11 in [95].

the  $t^{2k}$  choices, the overall probability  $\Pr[\text{SCI}(G'') = 1] < t^{2k}r^{k-1}/m = o(1)$ , for the parameter regime  $t^{2k} = o(m/\text{polylog}(m))$  and  $r = O(\log m)$ . Therefore,  $\mathcal{Q}$  outputs 1 with probability at most  $\text{err}(\Pi) + o(1) \leq \varepsilon + o(1)$ .

Thus far, we have a protocol  $\mathcal{Q}$  that outputs 1 with probability  $\alpha$  when  $\text{MPJ-MEET}(G^*) = 0$  and with probability  $\beta$  when  $\text{MPJ-MEET}(G^*) = 1$ , where  $\alpha \leq \varepsilon + o(1)$  and  $\beta \geq (2k)^{-2k}(1 - \varepsilon)$ . Recall that  $\varepsilon = (2k)^{-2k-2}$ , so  $\beta$  is bounded away from  $\alpha$ . Let  $\mathcal{Q}'$  be a protocol where we first toss an unbiased coin: if it lands heads, we output 0 with probability  $\delta := (\alpha + \beta)/2$  and 1 with probability  $1 - \delta$ ; if it lands tails, we simulate  $\mathcal{Q}$ . Then  $\mathcal{Q}'$  is a protocol for MPJ-MEET with error probability  $\frac{1}{2} - (\beta - \alpha)/4$ . By Lemma 2.1.8, this protocol must communicate  $\Omega(m^{1+1/(2k)}/k^{16} \log^{3/2} m)$  bits and so must  $\Pi$ .  $\square$

By a standard reduction from random-allocation communication protocols to random-order streaming algorithms, we obtain the following lower bound: the main result of this section.

**Theorem 2.1.9.** *For each constant  $p$ , a  $p$ -pass algorithm that solves SHORTPATH-DAG on  $n$ -vertex digraphs whose edges presented in a uniform random order, erring with probability at most  $1/p^{\Omega(p)}$  must use  $\Omega(n^{1+1/(2p+2)}/\log^{3/2} n)$  bits of space.*

*Consequently, similar lower bounds hold for the problems STCONN-DAG, ACYC, TOPO-SORT, FAS, and FAS-SIZE.*  $\square$

This paper is focused on *directed* graph problems. However, it is worth noting that a by-product of our generalization of the Guruswami–Onak bound to randomly ordered streams is that we also obtain the first random-order super-linear (in  $n$ ) lower bounds for two important *undirected* graph problems.

**Corollary 2.1.10.** *For each constant  $p$ ,  $n^{1+\Omega(1/p)}$  space is required to solve either of the following problems in  $p$  passes, erring with probability at most  $1/p^{\Omega(p)}$ , over a randomly ordered edge stream of an  $n$ -vertex undirected graph  $G$ :*

- *decide whether  $G$  contains a perfect matching;*
- *decide whether the distance between prespecified vertices  $v_s$  and  $v_t$  is at most  $2p + 2$ .*

**A Barrier Result.** Notably, Theorem 2.1.9 applies only to algorithms with a rather small error probability. This is inherent: allowing just a slightly larger error probability renders the problem solvable in semi-streaming space. This is shown in the result below, which should be read as a *barrier* result rather than a compelling algorithm.

**Proposition 2.1.11.** *Given a randomly ordered edge stream of a digraph  $G$ , the SHORTPATH-DAG problem on  $G$  can be solved using  $\tilde{O}(n)$  space and  $p$  passes, with error probability at most  $2/p!$ .*

*Proof.* Recall that we’re trying to decide whether or not  $G$  has a path of length at most  $(2p + 2)$  from  $v_s$  to  $v_t$ . The high-level idea is that thanks to the random ordering, a “Bellman–Ford” style algorithm that grows a forward path out of  $v_s$  and a backward path out of  $v_t$  is very likely to make more than one step of progress during *some* pass.

To be precise, we maintain arrays  $d_s$  and  $d_t$ , each indexed by  $V$ . Initialize the arrays to  $\infty$ , except that  $d_s[v_s] = d_t[v_t] = 0$ . During each pass, we use the following logic.

for each edge  $(x, y)$  in the stream:

if  $d_s[x] + d_t[y] \leq 2p + 1$ : output TRUE and halt

$d_s[y] \leftarrow \min(d_s[y], 1 + d_s[x])$

$d_t[x] \leftarrow \min(d_t[x], 1 + d_t[y])$

If we complete  $p$  passes without any output, then we output FALSE.

If  $G$  has no short enough path from  $v_s$  to  $v_t$ , this algorithm will always output FALSE. So let’s consider the other case, when there *is* a  $v_s$ – $v_t$  path  $\pi$  of length at most  $2p + 2$ . Let vertex  $z$  be the midpoint of  $\pi$ , breaking ties arbitrarily if needed. The subpaths  $[v_s, z]_\pi$  and  $[z, v_t]_\pi$  have lengths  $q$  and  $r$ , respectively, with  $q \leq p + 1$  and  $r \leq p + 1$ . Notice that if

our algorithm is allowed to run for  $q$  (resp.  $r$ ) passes, then  $d_s[z]$  (resp.  $d_t[z]$ ) will settle to its correct value. If both of them settle, then the algorithm correctly outputs TRUE. So, the only nontrivial case is when  $q, r \in \{p, p+1\}$ .

Let  $E_s$  be the event that the random ordering of the edges in the stream places the edges of  $[v_s, z]_\pi$  in the exact reverse order of  $\pi$ . Let  $E_t$  be the event that the random ordering places the edges of  $[z, v_t]_\pi$  in the exact same order as  $\pi$ . If  $E_s$  does not occur, then for some two consecutive edges  $(w, x), (x, y)$  on  $[v_s, z]_\pi$ , the stream puts  $(w, x)$  before  $(x, y)$ . Therefore, once  $d_s[w]$  settles to its correct value, the following pass will settle not just  $d_s[x]$ , but also  $d_s[y]$ ; therefore, after  $q-1 \leq p$  passes,  $d_s[z]$  is settled. Similarly, if  $E_t$  does not occur, then after  $r-1 \leq p$  passes,  $d_t[z]$  is settled. As noted above, if both of them settle, the algorithm correctly outputs TRUE.

Thus, the error probability  $\leq \Pr[E_s \vee E_t] \leq \Pr[E_s] + \Pr[E_t] = 1/q! + 1/r! \leq 2/p!$ , as required.  $\square$

### 2.1.5. Sink Finding in Tournaments

We now focus on tournaments, and begin with the sink-finding problem. A classical offline algorithm for TOPO-SORT is to repeatedly find a sink  $v$  in the input graph (which must exist in a DAG), prepend  $v$  to a growing list, and recurse on  $G \setminus v$ . Thus, SINK-FIND itself is a fundamental digraph problem. Obviously, SINK-FIND can be solved in a single pass using  $O(n)$  space by maintaining an “is-sink” flag for each vertex. Our results below show that for arbitrary order streams this is tight, even for tournament graphs.

In fact, we say much more. In  $p$  passes, on the one hand, the space bound can be improved to roughly  $O(n^{2/p})$ . On the other hand, any  $p$ -pass algorithm requires about  $\Omega(n^{1/p})$  space. While these bounds don’t quite match, they reveal the correct asymptotics for the number of passes required to achieve polylogarithmic space usage: namely,  $\Theta(\log n / \log \log n)$ .

In contrast, we show that if the stream is randomly ordered, then using  $\text{polylog}(n)$

space and a single pass is sufficient. Thus, we establish an exponential separation between the adversarial- and random-order streaming models.

**Arbitrary Order Sink Finding.** Let us first consider the setting where the stream order is adversarial.

**Theorem 2.1.12** (Multi-pass algorithm). *For all  $p$  with  $1 \leq p \leq \log n$ , there is a  $(2p - 1)$ -pass algorithm for SINK-FIND-T that uses  $O(n^{1/p} \log(3p))$  space and has failure probability at most  $1/3$ .*

*Proof.* Let the input digraph be  $G = (V, E)$ . For a set  $S \subseteq V$ , let  $\max S$  denote the vertex in  $S$  that has maximum in-degree. This can also be seen as the maximum vertex within  $S$  according to the total ordering defined by the edge directions.

Our algorithm proceeds as follows.

- *Initialization:* Set  $s = \lceil n^{1/p} \ln(3p) \rceil$ . Let  $S_1$  be a set of  $s$  vertices chosen randomly from  $V$ .
- For  $i = 1$  to  $p - 1$ :
  - *During pass  $2i - 1$ :* Find  $v_i = \max S_i$  by computing the in-degree of each vertex in  $S_i$ .
  - *During pass  $2i$ :* Let  $S_{i+1}$  be a set of  $s$  vertices chosen randomly from  $\{u : (v_i, u) \in E\}$ .
- *During pass  $2p - 1$ :* Find  $v_p = \max S_p$  by computing the in-degree of each vertex in  $S_p$ .

For the sake of analysis, consider the quantity  $\ell_i = |\{u : (v_i, u) \in E\}|$ . Note that, for each  $i \in [p]$ ,

$$\Pr [\ell_i > \ell_{i-1}/n^{1/p}] = (1 - 1/n^{1/p})^s \leq \frac{1}{3p}.$$

Thus, by the union bound,  $\ell_p = 0$  with probability at least  $1 - p/(3p) = 2/3$ . Note that  $\ell_p = 0$  implies that  $v_p$  is a sink.  $\square$

We turn to establishing a multi-pass lower bound. Our starting point for this is the *tree pointer jumping* problem  $\text{TPJ}_{k,t}$ , which is a communication game involving  $k$  players. To set up the problem, consider a complete ordered  $k$ -level  $t$ -ary tree  $T$ ; we consider its root  $z$  to be at level 0, the children of  $z$  to be at level 1, and so on. We denote the  $i$ -th child of  $y \in V(T)$  by  $y_i$ , the  $j$ -th child of  $y_i$  by  $y_{i,j}$ , and so on. Thus, each leaf of  $T$  is of the form  $z_{i_1, \dots, i_{k-1}}$  for some integers  $i_1, \dots, i_{k-1} \in [t]$ .

An instance of  $\text{TPJ}_{k,t}$  is given by a function  $\phi: V(T) \rightarrow [t]$  such that  $\phi(y) \in \{0, 1\}$  for each leaf  $y$ . The desired one-bit output is

$$\begin{aligned} \text{TPJ}_{k,t}(\phi) &:= g^{(k)}(z) = g(g(\dots g(z) \dots)), \text{ where} \\ g(y) &:= \begin{cases} \phi(y), & \text{if } y \text{ is a leaf,} \\ y_{\phi(y)}, & \text{otherwise.} \end{cases} \end{aligned} \quad (2.2)$$

For each  $j \in \{0, \dots, k-1\}$ , Player  $j$  receives the input values  $\phi(y)$  for each vertex  $y$  at level  $j$ . The players then communicate using at most  $k-1$  *rounds*, where a single round consists of one message from each player, speaking in the order Player  $k-1, \dots$ , Player 0. All messages are broadcast publicly (equivalently, written on a shared blackboard) and may depend on public random coins. The cost of a round is the *total* number of bits communicated in that round and the cost of a protocol is the *maximum*, over all rounds, of the cost of a round. The randomized complexity  $R^{k-1}(\text{TPJ}_{k,t})$  is the minimum cost of a  $(k-1)$ -round  $\frac{1}{3}$ -error protocol for  $\text{TPJ}_{k,t}$ .

Combining the lower bound approach of Chakrabarti et al. [56] with the improved round elimination analysis of Yehudayoff [171], we obtain the following lower bound on the randomized communication complexity of the problem.

**Theorem 2.1.13.**  $R^{k-1}(\text{TPJ}_{k,t}) = \Omega(t/k)$ . □

Based on this, we prove the following lower bound.

**Theorem 2.1.14** (Multi-pass lower bound). *Any streaming algorithm that solves SINK-FIND-T in  $p$  passes must use  $\Omega(n^{1/p}/p^2)$  space.*

*Proof.* We reduce from  $\text{TPJ}_{k,t}$ , where  $k = p + 1$ . We continue using the notations defined above. At a high level, we encode an instance of TPJ in the directions of edges in a tournament digraph  $G$ , where  $V(G)$  can be viewed as two copies of the set of leaves of  $T$ . Formally,

$$V(G) = \{\langle i_1, \dots, i_{k-1}, a \rangle : \text{each } i_j \in [t] \text{ and } a \in \{0, 1\}\}.$$

We assign each pair of distinct vertices  $u, v \in V(G)$  to a *level* in  $\{0, \dots, k-1\}$  as follows. Suppose that  $u = \langle i_1, \dots, i_k \rangle$  and  $v = \langle i'_1, \dots, i'_k \rangle$ . We assign  $\{u, v\}$  to level  $j-1$ , where  $j$  is the smallest index such that  $i_j \neq i'_j$ . Given an instance of  $\text{TPJ}_{k,t}$ , the players jointly create an instance of SINK-FIND-T as follows. For each  $j$  from  $k-1$  to  $0$ , in that order, Player  $j$  assigns directions for all pairs of vertices at level  $j$ , obtaining a set  $E_j$  of directed edges, and then appends  $E_j$  to a stream. The combined stream  $E_{k-1} \circ \dots \circ E_1 \circ E_0$  defines the tournament  $G$ . It remains to define each set  $E_j$  precisely.

The set  $E_{k-1}$  encodes the bits  $\phi(y)$  at the leaves  $y$  of  $T$  as follows.

$$E_{k-1} = \{(\langle i_1, \dots, i_{k-1}, 1-a \rangle, \langle i_1, \dots, i_{k-1}, a \rangle) \in V(G)^2 : \phi(z_{i_1, \dots, i_{k-1}}) = a\}, \quad (2.3)$$

Notice that if we ignore edge directions,  $E_{k-1}$  is a perfect matching on  $V(G)$ .

Now consider an arbitrary level  $j \in \{0, \dots, k-2\}$ . Corresponding to each vertex  $z_{i_1, \dots, i_{j-1}}$  at level  $j$  of  $T$ , we define the permutation  $\pi_{i_1, \dots, i_{j-1}} : [t] \rightarrow [t]$  thus:

$$(\pi_{i_1, \dots, i_{j-1}}(1), \dots, \pi_{i_1, \dots, i_{j-1}}(t)) = (1, \dots, \ell-1, \ell+1, \dots, t, \ell),$$

where  $\ell = \phi(z_{i_1, \dots, i_{j-1}})$ . (2.4)

Using this, we define  $E_j$  so as to encode the pointers at level  $j$  as follows.

$$E_j = \{(\langle i_1, \dots, i_{j-1}, i_j, \dots, i_k \rangle, \langle i_1, \dots, i_{j-1}, i'_j, \dots, i'_k \rangle) \in V(G)^2 : \pi_{i_1, \dots, i_{j-1}}^{-1}(i_j) < \pi_{i_1, \dots, i_{j-1}}^{-1}(i'_j)\}. \quad (2.5)$$

It should be clear that the digraph  $(V(G), E_0 \cup E_1 \cup \dots \cup E_{k-1})$  is a tournament. We argue that it is acyclic. Suppose, to the contrary, that  $G$  has a cycle  $\sigma$ . Let  $j \in \{0, \dots, k-2\}$  be the smallest-numbered level of an edge on  $\sigma$ . Then there exist  $h_1, \dots, h_{j-1}$  such that every vertex on  $\sigma$  is of the form  $\langle h_1, \dots, h_{j-1}, i_j, \dots, i_k \rangle$ . Let  $v^{(1)}, \dots, v^{(r)}$  be the vertices on  $\sigma$  whose outgoing edges belong to level  $j$ . For each  $q \in [r]$ , let  $v^{(q)} = \langle h_1, \dots, h_{j-1}, i_j^{(q)}, \dots, i_k^{(q)} \rangle$ . Let  $\hat{\pi} = \pi_{h_1, \dots, h_{j-1}}$ . According to eq. (2.5),

$$\hat{\pi}^{-1}(i_j^{(1)}) < \hat{\pi}^{-1}(i_j^{(2)}) < \dots < \hat{\pi}^{-1}(i_j^{(r)}) < \hat{\pi}^{-1}(i_j^{(1)}),$$

a contradiction.

It follows that  $G$  has a unique sink. Let  $v = \langle h_1, \dots, h_{k-1}, a \rangle \in V(G)$  be this sink. In particular, for each level  $j \in \{0, \dots, k-2\}$ , all edges in  $E_j$  involving  $v$  must be directed towards  $v$ . According to eq. (2.5), we must have  $\pi_{h_1, \dots, h_{j-1}}^{-1}(h_j) = t$ , i.e.,  $\pi_{h_1, \dots, h_{j-1}}(t) = h_j$ . By eq. (2.4), this gives  $\phi(z_{h_1, \dots, h_{j-1}}) = h_j$ . Next, by eq. (2.2), this gives  $g(z_{h_1, \dots, h_{j-1}}) = z_{h_1, \dots, h_j}$ . Instantiating this observation for  $j = 0, \dots, k-2$ , we have

$$z_{h_1} = g(z), \quad z_{h_1, h_2} = g(z_{h_1}), \quad \dots, \quad z_{h_1, \dots, h_{k-1}} = g(z_{h_1, \dots, h_{k-2}}),$$

$$\text{i.e., } z_{h_1, \dots, h_{k-1}} = g^{(k-1)}(z).$$

At this point  $h_1, \dots, h_{k-1}$  have been determined, leaving only two possibilities for  $v$ . We now use the fact that the sole edge in  $E_{k-1}$  involving  $v$  must be directed towards  $v$ . According to eq. (2.3),  $\phi(z_{h_1, \dots, h_{k-1}}) = a$ . Invoking eq. (2.2) again,  $a = \phi(g^{(k-1)}(z)) =$

$$g^{(k)}(z) = \text{TPJ}_{k,t}(\phi).$$

Thus, the players can read off the desired output  $\text{TPJ}_{k,t}(\phi)$  from the identity of the unique sink of the constructed digraph  $G$ . Notice that  $n := |V(G)| = 2t^{k-1}$ . It follows that a  $(k-1)$ -pass streaming algorithm for SINK-FIND-T that uses  $S$  bits of space solves  $\text{TPJ}_{k,t}$  in  $k-1$  rounds at a communication cost of  $kS$ . By Theorem 2.1.13, we have  $S = \Omega(t/k^2) = \Omega(n^{1/(k-1)}/k^2)$ .  $\square$

**Random Order Sink Finding.** Now, we shift to the random stream-order setting and show that it is possible to find the sink of an acyclic tournament in one pass while using only  $\text{polylog}(n)$  space. The algorithm we consider is as follows:

- *Initialization:* Let  $S$  be a random set of  $s = 200 \log n$  nodes.
- For  $i = 1$  to  $k := \log_2 \left( \frac{m}{200000n \log n} \right)$ :
  - Ingest the next  $c_i := 100 \cdot 2^i(n-1) \log n$  elements of the stream: For each  $v \in S$ , collect the set of edges  $S_v$  consisting of all outgoing edges; throw away  $S_v$  if it exceeds size  $220 \log n$
  - Pick any  $v \in S$ , such that  $|S_v| = (200 \pm 20) \log n$  and let  $S$  be the endpoints (other than  $v$ ) of the edges in  $S_v$
- Ingest the next  $m/1000$  elements: find  $P$  the set of vertices  $w$  such that there exists an edge  $uw$  for some  $u \in S$
- Ingest the remaining  $499m/500$  elements: Output any vertex in  $P$  with no outgoing edges.

Given a graph with a unique total ordering, we say a vertex  $u$  has rank  $rk(u) = r$  if it occurs in the  $r$ th position in this total ordering.

**Theorem 2.1.15.** *There is a single pass algorithm for SINK-FIND-T that uses  $O(\text{polylog } n)$  space and has failure probability at most  $1/3$  under the assumption that the data stream is*

randomly ordered.

*Proof.* We refer to the  $c_i$  elements used in the iteration  $i$  as the  $i$ th segment of the stream. For a node  $u$ , let  $X_{u,i}$  be the number of outgoing edges from  $u$  amongst the  $i$ th segment. The following claim follows from the Chernoff bound:

**Claim 2.1.16.** *With high probability, for all  $u$  with  $|rk(u) - n/2^i| \geq 0.2 \cdot n/2^i$  then*

$$|X_{u,i} - 200 \log n| > 0.1 \cdot 200 \log n .$$

*With high probability, for all  $u$  with  $|rk(u) - n/2^i| \leq 0.05 \cdot n/2^i$ , then*

$$|X_{u,i} - 200 \log n| < 0.1 \cdot 200 \log n .$$

It follows from the claim that if after processing the  $i$ th segment of the stream there exists a  $v$  such that  $|S_v| = (200 \pm 20) \log n$  then with high probability  $rk(u) = (1 \pm 0.2) \cdot n/2^i$ . We next need to argue that there exists such a  $v$ .

**Claim 2.1.17.** *With high probability, for every node  $u$  with  $rk(u) = (1 \pm 0.2) \cdot n/2^{i-1}$ , there exists an edge  $uv$  in the  $i$ th segment such that  $|rk(v) - n/2^i| \leq 0.05 \cdot n/2^i$ .*

*Proof.* There are at least  $0.01 \cdot n/2^i$  such edges. The probability that none of them exists in the  $i$ th segment is at most  $(1 - c_i/m)^{0.01 \cdot n/2^i} \leq 1/\text{poly}(n)$ .  $\square$

The above two claims allow us to argue by induction that we will have an element  $u$  with  $rk(u) = (1 \pm 0.2) \cdot n/2^i$  after the  $i$ th segment. At the end of the  $k$ th segment we have identified at least  $(200 - 20) \log n$  vertices where every rank is at most  $(1 + 0.2) \cdot n/2^k = O(\log n)$ . With probability at least  $1 - 1/\text{poly}(n)$  one of these vertices includes an edge to the sink amongst the  $(k + 1)$  segment and hence the sink is in  $P$  with high probability. There may be other vertices in  $P$  but the following claim shows that we will identify any false positives while processing the final  $499m/500$  elements of the stream.

**Claim 2.1.18.** *With probability at least  $1 - 1/499$ , there exists at least once outgoing edge from every node except the sink amongst the last  $499m/500$  elements of the stream*

*Proof of Claim.* The probability no outgoing edge from the an element of rank  $r > 0$  appears in the suffix of the stream is at most  $(1 - 499/500)^r$ . Hence, by the union bound the probability that there exists an element of rank  $r > 0$  without an outgoing edge is at most  $\sum_{r \geq 1} (1 - 499/500)^r = 1/499$ .  $\square$

This concludes the proof of Theorem 2.1.15.  $\square$

### 2.1.6. Feedback Arc Set in Tournaments

**Accurate, One Pass, but Slow Algorithm for FAS-T.** We shall now design an algorithm for FAS-T (that also solves FAS-SIZE-T) based on linear sketches for  $\ell_1$ -norm estimation. Recall that the  $\ell_1$ -norm of a vector  $\mathbf{x} \in \mathbb{R}^N$  is  $\|\mathbf{x}\|_1 = \sum_{i \in [N]} |x_i|$ . A  $d$ -dimensional  $\ell_1$ -sketch with accuracy parameter  $\varepsilon$  and error parameter  $\delta$  is a distribution  $\mathcal{S}$  over  $d \times N$  matrices, together with an estimation procedure  $\text{Est}: \mathbb{R}^d \rightarrow \mathbb{R}$  such that

$$\Pr_{S \leftarrow \mathcal{S}} \left[ (1 - \varepsilon)\|\mathbf{x}\|_1 \leq \text{Est}(S\mathbf{x}) \leq (1 + \varepsilon)\|\mathbf{x}\|_1 \right] \geq 1 - \delta.$$

Such a sketch is “stream friendly” if there is an efficient procedure to generate a given column of  $S$  and further,  $\text{Est}$  is efficient. Obviously, a stream friendly sketch leads to a space and time efficient algorithm for estimating  $\|\mathbf{x}\|_1$  given a stream of entrywise updates to  $\mathbf{x}$ . We shall use the following specialization of a result of Kane et al. [113].

**Fact 2.1.1** (Kane et al. [113]). *There is a stream friendly  $d$ -dimensional  $\ell_1$ -sketch with accuracy  $\varepsilon$  and error  $\delta$  that can handle  $N^{O(1)}$  many  $\pm 1$ -updates to  $\mathbf{x} \in \mathbb{R}^N$ , with each update taking  $O(\varepsilon^{-2} \log \varepsilon^{-1} \log \delta^{-1} \log N)$  time, with  $d = O(\varepsilon^{-2} \log \delta^{-1})$ , and with entries of the sketched vector fitting in  $O(\log N)$  bits.*

**Theorem 2.1.19.** *There is a one-pass algorithm for FAS-T that uses  $O(\varepsilon^{-2}n \log^2 n)$  space and returns a  $(1 + \varepsilon)$ -approximation with probability at least  $\frac{2}{3}$ , but requires exponential post-processing time.*

*Proof.* Identify the vertex set of the input graph  $G = (V, E)$  with  $[n]$  and put  $N = \binom{n}{2}$ . We index vectors  $\mathbf{z}$  in  $\mathbb{R}^N$  as  $z_{uv}$ , where  $1 \leq u < v \leq n$ . Define a vector  $\mathbf{x} \in \{0, 1\}^N$  based on  $G$  and vectors  $\mathbf{y}^\pi \in \{0, 1\}^N$  for each permutation  $\pi: [n] \rightarrow [n]$  using indicator variables as follows.

$$x_{uv} = \mathbb{1}\{(u, v) \in E\}, \quad y_{uv}^\pi = \mathbb{1}\{\pi(u) < \pi(v)\}.$$

A key observation is that the  $uv$ -entry of  $\mathbf{x} - \mathbf{y}^\pi$  is nonzero iff the edge between  $u$  and  $v$  is a back edge of  $G$  according to the ordering  $\pi$ . Thus,  $|B_G(\pi)| = \|\mathbf{x} - \mathbf{y}^\pi\|_1$ .

Our algorithm processes the graph stream by maintaining an  $\ell_1$ -sketch  $S\mathbf{x}$  with accuracy  $\varepsilon/3$  and error  $\delta = 1/(3 \cdot n!)$ . By Fact 2.1.1, this takes  $O(\varepsilon^{-2}n \log^2 n)$  space and  $O(\varepsilon^{-2} \log \varepsilon^{-1}n \log^2 n)$  time per edge.

In post-processing, the algorithm considers all  $n!$  permutations  $\pi$  and, for each of them, computes  $S(\mathbf{x} - \mathbf{y}^\pi) = S\mathbf{x} - S\mathbf{y}^\pi$ . It thereby recovers an estimate for  $\|\mathbf{x} - \mathbf{y}^\pi\|_1$  and finally outputs the ordering  $\pi$  that minimizes this estimate. By a union bound, the probability that every estimate is  $(1 \pm \varepsilon/3)$ -accurate is at least  $1 - n! \cdot \delta = 2/3$ . When this happens, the output ordering provides a  $(1 + \varepsilon)$ -approximation to FAS-T by our key observation above.  $\square$

Despite its “brute force” feel, the above algorithm is essentially optimal, both in its space usage (unconditionally) and its post-processing time (in a sense we shall make precise later). We address these issues in Section 2.1.6.

**Multiple Passes: FAS-T in Polynomial Time.** For a more time-efficient streaming algorithm, we design one based on the KWIKSORT algorithm of Ailon et al. [7]. This (non-streaming) algorithm operates as follows on a tournament  $G = (V, E)$ .

- Choose a random ordering of the vertices:  $v_1, v_2, \dots, v_n$ .

- Vertex  $v_1$  partitions  $V$  into two sub-problems  $\{u : (u, v_1) \in E\}$  and  $\{w : (v_1, w) \in E\}$ . At this point we know the exact place of  $v_1$  in the ordering.
- Vertex  $v_2$  further partitions one of these sub-problems. Proceeding in this manner, after  $v_1, v_2, \dots, v_i$  are considered, there are  $i + 1$  sub-problems.
- Continue until all  $n$  vertices are ordered.

When  $v_i$  is being used to divide a sub-problem we refer to it as a *pivot*.

**Emulating KWIKSORT in the Data Stream Model.** We will emulate KWIKSORT in  $p$  passes over the data stream. In each pass, we will consider the action of multiple pivots. Partition  $v_1, \dots, v_n$  into  $p$  groups  $V_1, \dots, V_p$ , where  $V_1 = \{v_1, \dots, v_{cn^{1/p} \log n}\}$ ,  $V_2$  consists of the next  $cn^{2/p} \log n$  vertices in the sequence, and  $V_j$  contains  $cn^{j/p} \log n$  vertices coming after  $V_{j-1}$ . Here  $c$  is a sufficiently large constant. At the end of pass  $j + 1$ , we want to emulate the effect of pivots in  $V_{j+1}$  on the sub-problems resulting from considering pivots in  $V_1$  through  $V_j$ . In order to do that, in pass  $j + 1$  for each vertex  $v \in V_{j+1}$  we store all edges between  $v$  and vertices in the same sub-problem as  $v$ , where the sub-problems are defined at the end of pass  $j$ .

The following combinatorial lemma plays a key role in analyzing this algorithm's space usage.

**Lemma 2.1.20 (Mediocrity Lemma).** *In an  $n$ -vertex tournament, if we pick a vertex  $v$  uniformly at random, then  $\Pr[\varepsilon n < d_{\text{in}}(v) < (1 - \varepsilon)n] \geq 1 - 4\varepsilon$ .*

*Similarly,  $\Pr[\varepsilon n < d_{\text{out}}(v) < (1 - \varepsilon)n] \geq 1 - 4\varepsilon$ . In particular, with probability at least  $1/3$ ,  $v$  has in/out-degree between  $n/6$  and  $5n/6$ .<sup>5</sup>*

<sup>5</sup>The Mediocrity Lemma is tight: consider sets of vertices  $A, B, C$  where  $|A| = |C| = 2\varepsilon n$  and  $|B| = (1 - 4\varepsilon)n$ . Edges on  $B$  do not form any directed cycles. Subgraphs induced by  $A$  and  $C$  are balanced, i.e., the in-degree equals the out-degree of every vertex (where degrees here are considered within the subgraph). All other edges are directed from  $A$  to  $B$ , from  $B$  to  $C$ , or from  $A$  to  $C$ . Then vertices with in/out-degrees between  $\varepsilon n$  and  $(1 - \varepsilon)n$  are exactly the vertices in  $B$ , and a random vertex belongs to this set with probability  $1 - 4\varepsilon$ .

*Proof.* Let  $H$  be a set of vertices of in-degree at least  $(1 - \varepsilon)n$ . Let  $h = |H|$ . On the one hand,  $\sum_{v \in H} d_{\text{in}}(v) \geq (1 - \varepsilon)nh$ . On the other hand, the edges that contribute to the in-degrees of vertices in  $H$  have both endpoints in  $H$  or one endpoint in  $H$  and one in  $V \setminus H$ . The number of such edges is

$$\sum_{v \in H} d_{\text{in}}(v) \leq \binom{h}{2} + h(n - h) = \frac{1}{2}(2nh - h^2 - h).$$

Therefore,  $(2nh - h^2 - h)/2 \geq (1 - \varepsilon)nh$ . This implies  $h < 2\varepsilon n$ .

Thus, the number of vertices with in-degree at least  $(1 - \varepsilon)n$  (and out-degree at most  $\varepsilon n$ ) is  $h < 2\varepsilon n$ . By symmetry, the number of vertices with out-degree at least  $(1 - \varepsilon)n$  (and in-degree at most  $\varepsilon n$ ) is also less than  $2\varepsilon n$ . Thus, the probability a random vertex has in/out-degree between  $\varepsilon n$  and  $(1 - \varepsilon)n$  is  $(n - 2h)/n > (n - 2 \cdot 2\varepsilon n)/n = 1 - 4\varepsilon$ .  $\square$

**Space Analysis.** Let  $M_j$  be the maximum size of a sub-problem after pass  $j$ . The number of edges collected in pass  $j + 1$  is then at most  $M_j |V_{j+1}|$ . By Lemma 2.1.21 (below), this is at most  $cn^{1+1/p} \log n$ . Once the post-processing of pass  $j + 1$  is done, the edges collected in that pass can be discarded.

**Lemma 2.1.21.** *With high probability,  $M_j \leq n^{1-j/p}$  for all  $j$ .*

*Proof.* Let  $M_j^v$  denote the size of the sub-problem that contains  $v$ , after the  $j$ th pass. We shall prove that, for each  $v$ ,  $\Pr[M_j^v > n^{1-j/p}] \leq 1/n^{10}$ . The lemma will then follow by a union bound.

Take a particular vertex  $v$ . If, before the  $j$ th pass, we already have  $M_{j-1}^v \leq n^{1-j/p}$ , there is nothing to prove. So assume that  $M_{j-1}^v > n^{1-j/p}$ . Call a pivot “good” if it reduces the size of the sub-problem containing  $v$  by a factor of at least  $5/6$ . A random pivot falls in the same sub-problem as  $v$  with probability at least  $n^{1-j/p}/n$ ; when this happens, by the Mediocrity Lemma, the probability that the pivot is good is at least  $1/3$ . Overall, the probability that the pivot is good is at least  $n^{-j/p}/3$ .

In the  $j$ th pass, we use  $cn^{j/p} \log n$  pivots. If at least  $\log_{6/5} n$  of them are good, we definitely have  $M_j^v \leq n^{1-j/p}$ . Thus, by a Chernoff bound, for a sufficiently large  $c$ , we have

$$\Pr [M_j^v > n^{1-j/p}] \leq \Pr [\text{Bin}(cn^{j/p} \log n, n^{-j/p}/3) < \log_{6/5} n] \leq 1/n^{10}. \quad \square$$

**Theorem 2.1.22.** *There exists a polynomial time  $p$ -pass data stream algorithm using  $\tilde{O}(n^{1+1/p})$  space that returns a 3-approximation (in expectation) for FAS-T. In particular, there is a  $\log n$ -pass semi-streaming algorithm for 3-approximate FAS-T.*

*Proof.* The pass/space tradeoff follows from Lemma 2.1.21 and the discussion above it; the approximation factor follows directly from the analysis of Ailon et al. [7]. The second part follows by substituting  $p = \log n$ .  $\square$

**A Space Lower Bound.** Both our one-pass algorithm and the  $O(\log n)$ -pass instantiation of our multi-pass algorithm use at least  $\Omega(n)$  space. For FAS-SIZE-T, where the desired output is a just a number, it is reasonable to ask whether  $o(n)$ -space solutions exist. We now prove that they do not.

**Proposition 2.1.23.** *Solving ACYC-T is possible in one pass and  $O(n \log n)$  space. Meanwhile, any  $p$ -pass solution requires  $\Omega(n/p)$  space.*

*Proof.* For the upper bound, we maintain the in-degrees of all vertices in the input graph  $G$ . Since  $G$  is a tournament, the set of in-degrees is exactly  $\{0, 1, \dots, n-1\}$  iff the input graph is acyclic.

For the lower bound, we reduce from  $\text{DISJ}_{N, N/3}$ . Alice and Bob construct a tournament  $T$  on  $n = 7N/3$  vertices, where the vertices are labeled  $\{v_1, \dots, v_{2N}, w_1, \dots, w_{N/3}\}$ . Alice, based on her input  $\mathbf{x}$ , adds edges  $(v_{2i}, v_{2i-1})$  for each  $i \in \mathbf{x}$ . For each remaining pair  $(i, j) \in [2N] \times [2N]$  with  $i < j$ , she adds the edge  $(v_i, v_j)$ . Let  $a_1 < \dots < a_{N/3}$  be the

sorted order of the elements in Bob's set  $\mathbf{y}$ . For each  $k = a_\ell \in \mathbf{y}$ , Bob defines the alias  $v_{2N+k} = w_\ell$  and then adds the edges

$$E_k = \{(v_i, v_{2N+k}) : 1 \leq i \leq 2k-1\} \cup \{(v_{2N+k}, v_j) : 2k \leq j \leq 2N\}.$$

Finally, he adds the edges  $\{(w_i, w_j) : 1 \leq i < j \leq N/3\}$ . This completes the construction of  $T$ .

We claim that the tournament  $T$  is acyclic iff  $\mathbf{x} \cap \mathbf{y} = \emptyset$ . The “only if” part is direct from construction, since if  $\mathbf{x}$  and  $\mathbf{y}$  intersect at some index  $k \in [N]$ , we have the directed cycle  $(v_{2k}, v_{2k-1}, v_{2N+k}, v_{2k})$ . For the “if” part, let  $\sigma$  be the ordering  $(v_1, \dots, v_{2N})$  and let  $T' = \text{ToU}(\sigma)$ , as defined in Section 2.1.3. We show how to modify  $\sigma$  into a topological ordering of  $T$ , proving that  $T$  is acyclic. Observe that, by construction, the tournament  $T \setminus \{w_1, \dots, w_{N/3}\}$  can be obtained from  $T'$  by flipping only the edges  $(v_{2i-1}, v_{2i})$  for each  $i \in \mathbf{x}$ . Each time we perform such an edge flip, we modify the topological ordering of  $T'$  by swapping the associated vertices of the edge. The resultant ordering would still be topological as the vertices were consecutive in the ordering before the flip. Thus, after performing these swaps, we get a topological ordering of  $T \setminus \{w_1, \dots, w_{N/3}\}$ . Now, consider some  $k \in \mathbf{y}$ . Since  $\mathbf{x} \cap \mathbf{y} = \emptyset$ ,  $k \notin \mathbf{x}$  and so,  $v_{2k}$  succeeds  $v_{2k-1}$  in this ordering, just as in  $\sigma$ , since we never touched these two vertices while performing the swaps. Thus, for each such  $k$ , we can now insert  $v_{2N+k}$  between  $v_{2k-1}$  and  $v_{2k}$  in the ordering and obtain a topological ordering of  $T$ . This proves the claim.

Thus, given a  $p$ -pass solution to ACYC-T using  $s$  bits of space, we obtain a protocol for  $\text{DISJ}_{N, N/3}$  that communicates at most  $(2p-1)s$  bits. By Fact 1.3.3,  $(2p-1)s = \Omega(N) = \Omega(n)$ , i.e.,  $s = \Omega(n/p)$ .  $\square$

**Theorem 2.1.24.** *A  $p$ -pass multiplicative approximation for FAS-SIZE-T requires  $\Omega(n/p)$  space.*

*Proof.* This is immediate from Observation 2.1.1 and proposition 2.1.23.  $\square$

**A Query Lower Bound.** Let us now consider the nature of the post-processing performed by our one-pass FAS-T algorithm in Section 2.1.6. During its streaming pass, that algorithm builds an *oracle* based on  $G$  that, when queried on an ordering  $\sigma$ , returns a fairly accurate estimate of  $|B_G(\sigma)|$ . It proceeds to query this oracle  $n!$  times to find a good ordering. This raises the question: is there a more efficient way to exploit the oracle that the algorithm has built? A similar question was asked in Bateni et al. [34] in the context of using sketches for the maximum coverage problem.

Were the oracle *exact*—i.e., on input  $\sigma$  it returned  $|B_G(\sigma)|$  exactly—then two queries to the oracle would determine which of  $(i, j)$  and  $(j, i)$  was an edge in  $G$ . It follows that  $O(n \log n)$  queries to such an exact oracle suffice to solve FAS-T and FAS-SIZE-T. However, what we actually have is an  $\varepsilon$ -oracle, defined as one that, on query  $\sigma$ , returns  $\hat{\beta} \in \mathbb{R}$  such that  $(1 - \varepsilon)|B_G(\sigma)| \leq \hat{\beta} \leq (1 + \varepsilon)|B_G(\sigma)|$ . We shall show that an  $\varepsilon$ -oracle cannot be exploited efficiently: a randomized algorithm will, with high probability, need exponentially many queries to such an oracle to solve either FAS-T or FAS-SIZE-T.

To prove this formally, we consider two distributions on  $n$ -vertex tournaments, defined next.

**Definition 2.1.25.** Let  $\mathcal{D}_{\text{yes}}, \mathcal{D}_{\text{no}}$  be distributions on tournaments on  $[n]$  produced as follows. To produce a sample from  $\mathcal{D}_{\text{yes}}$ , pick a permutation  $\pi$  of  $[n]$  uniformly at random; output  $\text{Tou}(\pi)$ . To produce a sample from  $\mathcal{D}_{\text{no}}$ , for each  $i, j$  with  $1 \leq i < j \leq n$ , independently at random, include edge  $(i, j)$  with probability  $\frac{1}{2}$ ; otherwise include edge  $(j, i)$ .

Let  $\sigma$  be an ordering of  $[n]$ . By linearity of expectation, if  $T$  is sampled from either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$ ,

$$\mathbb{E}|B_T(\sigma)| = m := \frac{1}{2} \binom{n}{2}.$$

In fact, we can say much more.

**Lemma 2.1.26.** *There is a constant  $c$  such that, for all  $\varepsilon > 0$ , sufficiently large  $n$ , a fixed*

ordering  $\sigma$  on  $[n]$ , and random  $T$  drawn from either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$ ,

$$\Pr [(1 - \varepsilon)m < |B_T(\sigma)| < (1 + \varepsilon)m] \geq 1 - 2^{-c\varepsilon^2 n}.$$

*Proof.* When  $T \leftarrow \mathcal{D}_{\text{no}}$ , the random variable  $|B_T(\sigma)|$  has binomial distribution  $\text{Bin}(2m, \frac{1}{2})$ , so the claimed bound is immediate.

Let  $T \leftarrow \mathcal{D}_{\text{yes}}$ . Partition the edges of the tournament into perfect matchings  $M_1, \dots, M_{n-1}$ . For each  $i \in [n - 1]$ , let  $X_i$  be the number of back edges of  $T$  involving  $M_i$ , i.e.,

$$X_i = |\{(u, v) \in M_i : \text{either } (u, v) \in B_T(\sigma) \text{ or } (v, u) \in B_T(\sigma)\}|.$$

Notice that  $X_i \sim \text{Bin}(n/2, \frac{1}{2})$ , whence

$$\Pr [(1 - \varepsilon)n/4 < X_i < \frac{1}{2}(1 + \varepsilon)n/4] \geq 1 - 2^{-b\varepsilon^2 n},$$

for a certain constant  $b$ . By a union bound, the probability that *all* of the  $X_i$ s are between these bounds is at least  $1 - (n - 1)2^{-b\varepsilon^2 n} \geq 1 - 2^{-c\varepsilon^2 n}$ , for suitable  $c$ . When this latter event happens, we also have  $(1 - \varepsilon)m < |B_T(\sigma)| = \frac{1}{2} \sum_{i=1}^{n-1} X_i < (1 + \varepsilon)m$ .  $\square$

We define a  $(q, \varepsilon)$ -query algorithm for a problem  $P$  to be one that access an input digraph  $G$  solely through queries to an  $\varepsilon$ -oracle and, after at most  $q$  such queries, outputs its answer to  $P(G)$ . We require this answer to be correct with probability at least  $\frac{2}{3}$ .

Now consider the particular oracle  $\mathcal{O}_{T, \varepsilon}$ , describing an  $n$ -vertex tournament  $T$ , that behaves as follows when queried on an ordering  $\sigma$ .

- If  $(1 - \varepsilon/2)m < |B_T(\sigma)| < (1 + \varepsilon/2)m$ , then return  $m$ .
- Otherwise, return  $|B_T(\sigma)|$ .

Clearly,  $\mathcal{O}_{T, \varepsilon}$  is an  $\varepsilon$ -oracle. The intuition in the next two proofs is that this oracle makes life difficult by seldom providing useful information.

**Proposition 2.1.27.** *Every  $(q, \varepsilon)$ -query algorithm for TOPO-SORT-T makes  $\exp(\Omega(\varepsilon^2 n))$  queries.*

*Proof.* WLOG, consider a  $(q, \varepsilon)$ -query algorithm,  $\mathcal{A}$ , that makes exactly  $q$  queries, the last of which is its output. Using Yao's minimax principle, fix  $\mathcal{A}$ 's random coins, obtaining a deterministic  $(q, \varepsilon)$ -query algorithm  $\mathcal{A}'$  that succeeds with probability  $\geq \frac{2}{3}$  on a random tournament  $T \leftarrow \mathcal{D}_{\text{yes}}$ . Let  $\sigma_1, \dots, \sigma_q$  be the sequence of queries that  $\mathcal{A}'$  makes when the answer it receives from the oracle to each of  $\sigma_1, \dots, \sigma_{q-1}$  is  $m$ .

Suppose that the oracle supplied to  $\mathcal{A}'$  is  $\mathcal{O}_{T, \varepsilon}$ . Let  $\mathcal{E}$  be the event that  $\mathcal{A}'$ 's query sequence is  $\sigma_1, \dots, \sigma_q$  and it receives the response  $m$  to each of these queries. For a particular  $\sigma_i$ ,

$$\Pr[\mathcal{O}_{T, \varepsilon}(\sigma_i) = m] = \Pr[(1 - \varepsilon/2)m < |B_T(\sigma_i)| < (1 + \varepsilon/2)m] \geq 1 - 2^{-b\varepsilon^2 n}$$

for a suitable constant  $b$ , by Lemma 2.1.26. Thus, by a union bound,  $\Pr[\mathcal{E}] \geq 1 - q2^{-b\varepsilon^2 n}$ .

When  $\mathcal{E}$  occurs,  $\mathcal{A}'$  must output  $\sigma_q$ , but  $\mathcal{E}$  itself implies that  $|B_T(\sigma_q)| \neq 0$ , so  $\mathcal{A}'$  errs. Thus, the success probability of  $\mathcal{A}'$  is at most  $1 - \Pr[\mathcal{E}] \leq q2^{-b\varepsilon^2 n}$ . Since this probability must be at least  $\frac{2}{3}$ , we need  $q \geq \frac{2}{3} \cdot 2^{b\varepsilon^2 n} = \exp(\Omega(\varepsilon^2 n))$ .  $\square$

**Proposition 2.1.28.** *Every  $(q, \varepsilon)$ -query algorithm for ACYC-T makes  $\exp(\Omega(\varepsilon^2 n))$  queries.*

*Proof.* We proceed similarly to Proposition 2.1.27, except that we require the deterministic  $(q, \varepsilon)$ -query algorithm  $\mathcal{A}'$  to succeed with probability at least  $\frac{2}{3}$  on a random  $T \leftarrow \frac{1}{2}(\mathcal{D}_{\text{yes}} + \mathcal{D}_{\text{no}})$ . We view  $T$  as being chosen in two stages: first, we pick  $Z \in_R \{\text{yes}, \text{no}\}$  uniformly at random, then we pick  $T \leftarrow \mathcal{D}_Z$ .

Define  $\sigma_1, \dots, \sigma_q$  and  $\mathcal{E}$  as before. So  $\Pr[\mathcal{E}] \geq 1 - q2^{-b\varepsilon^2 n}$ . When  $\mathcal{E}$  occurs,  $\mathcal{A}'$  must output some fixed answer, either “yes” or “no.” We consider these cases separately.

Suppose that  $\mathcal{A}'$  outputs “no,” declaring that  $T$  is not acyclic. Then  $\mathcal{A}'$  errs whenever

$Z = \text{yes}$  and  $\mathcal{E}$  occurs. The probability of this is at least  $\frac{1}{2} - q2^{-b\varepsilon^2 n}$ , but it must be at most  $\frac{1}{3}$ , requiring  $q = \exp(\Omega(\varepsilon^2 n))$ .

Suppose that  $\mathcal{A}'$  outputs “yes” instead. Then it errs when  $Z = \text{no}$ ,  $T$  is cyclic, and  $\mathcal{E}$  occurs. Since

$$\Pr[T \text{ acyclic} \mid Z = \text{no}] = n!/2^{\binom{n}{2}} = \exp(-\Omega(n^2)),$$

we have  $\frac{1}{3} \geq \Pr[\mathcal{A}' \text{ errs}] \geq \frac{1}{2} - \exp(-\Omega(n^2)) - q2^{-b\varepsilon^2 n}$ , requiring  $q = \exp(\Omega(\varepsilon^2 n))$ .  $\square$

**Theorem 2.1.29.** *A  $(q, \varepsilon)$ -query algorithm that gives a multiplicative approximation for either FAS-T or FAS-SIZE-T must make  $q = \exp(\Omega(\varepsilon^2 n))$  queries.*

*Proof.* This is immediate from Observation 2.1.1 and propositions 2.1.27 and 2.1.28.  $\square$

### 2.1.7. Topological Ordering in Random Graphs

We present results for computing a topological ordering of  $G \sim \text{PlantDAG}_{n,q}$  (see Definition 2.1.2). We first present an  $O(\log n)$ -pass algorithm using  $\tilde{O}(n^{4/3})$  space. We then present a one-pass algorithm that uses  $\tilde{O}(n^{3/2})$  space and requires the assumption that the stream is in random order.

**Arbitrary Order Algorithm.** Here, we present two different algorithms. The first is appropriate when  $q$  is large whereas the second is appropriate when  $q$  is small. Combining these algorithms and considering the worst case value of  $q$  yields the algorithm using  $\tilde{O}(n^{4/3})$  space.

**Algorithm for large  $q$ .** The basic approach is to emulate QuickSort. We claim that we can find the relationship between any vertex  $u$  among  $n$  vertices and a predetermined vertex  $v$  using three passes and  $O(n + q^{-3} \log n)$  space. Assuming this claim, we can sort in  $O(\log(q^2 n))$  passes and  $\tilde{O}(n/q)$  space: we recursively partition the vertices and suppose at the end of a phase we have sub-problems of sizes  $n_1, n_2, n_3, \dots$ . Any sub-problem with at least  $1/q^2$  vertices is then sub-divided by picking  $\Theta(\log n)$  random pivots

(with replacement) within the sub-problems using the aforementioned three pass algorithm. There are at most  $q^2 n$  such sub-problems. Hence, the total space required partition all the sub-problems in this way is at most

$$O \left( \log n \sum_{i=1}^{q^2 n} (n_i + q^{-3} \log n) \right) = O(nq^{-1} \log^2 n) .$$

Note that the size of every sub-problem decreases by a factor at least 2 at each step with high probability and hence after  $\log(q^2 n)$  iterations, all sub-problems have at most  $1/q^2$  vertices. Furthermore, each vertex degree is  $O(1/q \cdot \log n)$  in each sub-problem. Hence, the entire remaining instance can be stored using  $O(n/q \cdot \log n)$  space.

It remains to prove our three-pass claim. For this, we define the following families of sets:

$$L_i = \{u : \exists u\text{-to-}v \text{ path of length } \leq i\} , \quad R_i = \{u : \exists v\text{-to-}u \text{ path of length } \leq i\} .$$

We shall call an edge *critical* if it lies on a directed Hamiltonian path of length  $n - 1$  in a DAG. Using two passes and  $O(n \log n)$  space we can identify  $L_2$  and  $R_2$  using  $O(n \log n)$  space. Let  $U$  be the set of vertices not contained in  $L_2 \cup R_2$ . The following lemma (which can be proved via Chernoff bounds) establishes that  $L_2 \cup R_2$  includes most of the vertices of the graph with high probability.

**Lemma 2.1.30.** *With high probability,  $|U| = O(q^{-2} \log n)$ .*

In a third pass, we store every edge between vertices in  $U$  and also compute  $L_3$  and  $R_3$ . Computing  $L_3$  and  $R_3$  requires only  $O(n \log n)$  space. There is an edge between each pair of vertices in  $U$  with probability  $q$  and hence, the expected number of edges between vertices in  $U$  is at most  $q|U|^2 = O(q^{-3} \log^2 n)$ . By an application of the Chernoff Bound, this bound also holds w.h.p. Note that  $L_3, R_3$ , and the edges within  $U$  suffice to determine

whether  $u \in L_\infty$  or  $u \in R_\infty$  for all  $u$ . To see this first suppose  $u \in L_\infty$  and that  $(u, w)$  is the critical edge on the directed path from  $u$  to  $v$ . Either  $w \in L_2$  and therefore we deduce  $u \in L_3$ ; or  $u \in L_2$ ; or  $u \notin L_2$  and  $w \notin L_2$  and we therefore store the edge  $(u, w)$ .

This establishes the following lemma.

**Lemma 2.1.31.** *There is a  $O(\log n)$ -pass,  $\tilde{O}(n/q)$ -space algorithm for TOPO-SORT on a random input graph  $G \sim \text{PlantDAG}_{n,q}$ .  $\square$*

**Algorithm for small  $q$ .** We use only two passes. In the first pass, we compute the in-degree of every vertex. In the second, we store all edges between vertices where the in-degrees differ by at most  $3\sqrt{cnq \cdot \ln n}$  where  $c > 0$  is a sufficiently large constant.

**Lemma 2.1.32.** *There is a two-pass,  $\tilde{O}(n^{3/2}\sqrt{q})$ -space algorithm for TOPO-SORT on a random input graph  $G \sim \text{PlantDAG}_{n,q}$ .*

*Proof.* We show that, with high probability, the above algorithm collects all critical edges and furthermore only collects  $\tilde{O}(n^{3/2}\sqrt{q})$  edges in total. Let  $u$  be the element of rank  $r_u$ . Note that  $d_{\text{in}}(u)$  has distribution  $1 + \text{Bin}(r_u - 2, q)$ . Let  $X_u = d_{\text{in}}(u) - 1$ . By an application of the Chernoff Bound,

$$\Pr \left[ |X_u - (r_u - 2)q| \geq \sqrt{c(r_u - 2)q \ln n} \right] \leq 1/\text{poly}(n).$$

Hence, w.h.p.,  $r_u = 2 + X_u/q \pm \sqrt{cn/q \cdot \ln n}$  for all vertices  $u$ . Therefore, if  $(u, v)$  is critical, then

$$|X_u - X_v| \leq |X_u - (r_u - 2)q| + |(r_u - 2)q - (r_v - 2)q| + |X_v - (r_v - 2)q| \leq 3\sqrt{cnq \cdot \ln n}.$$

This ensures that the algorithm collects all critical edges. For the space bound, we first

observe that for an arbitrary pair of vertices  $u$  and  $v$ , if  $|X_u - X_v| \leq 3\sqrt{cnq \cdot \ln n}$  then

$$|r_u - r_v| \leq |X_u - X_v|/q + 2\sqrt{cn/q \cdot \ln n} \leq 8\sqrt{cn/q \cdot \ln n}.$$

Hence, we only store an edge between vertex  $u$  and vertices whose rank differs by at most  $8\sqrt{cn/q \cdot \ln n}$ . Since edges between such vertices are present with probability  $q$ , the expected number of edges stored incident to  $u$  is  $8\sqrt{cnq \cdot \ln n}$  and is  $O(\sqrt{nq \cdot \ln n})$  by an application of the Chernoff bounds. Across all vertices this means the number of edges stored is  $O(n^{3/2}\sqrt{q \cdot \ln n})$  as claimed.  $\square$

Combining Lemma 2.1.31 and Lemma 2.1.32 yields the main theorem of this section.

**Theorem 2.1.33.** *There is an  $O(\log n)$ -pass algorithm for TOPO-SORT on a random input  $G \sim \text{PlantDAG}_{n,q}$  that uses  $\tilde{O}(\min(n/q, n^{3/2}\sqrt{q}))$  space. For the worst-case over  $q$ , this is  $\tilde{O}(n^{4/3})$ .*  $\square$

**Random Order Algorithm.** The transitive reduction of a DAG  $G = (V, E)$  is the minimal subgraph  $G^{\text{red}} = (V, E')$  such that, for all  $u, v \in V$ , if  $G$  has a  $u$ -to- $v$  path, then so does  $G^{\text{red}}$ . So if  $G$  has a Hamiltonian path,  $G^{\text{red}}$  is this path.

The one-pass algorithm assuming a random ordering of the edges is simply to maintain  $G^{\text{red}}$  as  $G$  is streamed in, as follows. Let  $S$  be initially empty. For each edge  $(u, v)$  in the stream, we add  $(u, v)$  to  $S$  and then remove all edges  $(u', v')$  where there is a  $u'$ -to- $v'$  path among the stored edges.

**Theorem 2.1.34.** *There is a one-pass  $\tilde{O}(\max_{\hat{q} \leq q} \min\{n/\hat{q}, n^2\hat{q}\})$ -space algorithm for TOPO-SORT on inputs  $G \sim \text{PlantDAG}_{n,q}$  presented in random order. In the worst case this space bound is  $\tilde{O}(n^{3/2})$ .*

*Proof.* Consider the length- $T$  prefix of the stream where the edges of  $G$  are presented in random order. It will be convenient to write  $T = n^2\hat{q}$ . We will argue that the number of

edges in the transitive reduction of this prefix is  $O(\min\{n/\hat{q}, n^2\hat{q}\})$  with high probability; note the bound  $n^2\hat{q}$  follows trivially because the transitive reduction has at most  $T$  edges. The result then follows by taking the maximum over all prefixes.

We say an edge  $(u, v)$  of  $G$  is *short* if the difference between the ranks is  $r_v - r_u \leq \tau := c\hat{q}^{-2} \log n$  where  $c$  is some sufficiently large constant. An edge that is not short is defined to be *long*. Let  $S$  be the number of short edges in  $G$  and let  $M$  be the total number of edges in  $G$ . Note that  $\mathbb{E}[S] \leq (n-1) + q\tau n$  and  $\mathbb{E}[M] = (n-1) + q\binom{n-1}{2}$ . By the Chernoff bound,  $S \leq 2q\tau n$  and  $n^2q/4 \leq M \leq n^2q$  with high probability. Furthermore, the number of short edges in the prefix is expected to be  $T \cdot S/M$  and, with high probability, is at most

$$2T \cdot S/M \leq \frac{4Tq\tau n}{n^2q/4} = 16cn/\hat{q} \cdot \log n.$$

Now consider how many long edges are in the transitive reduction of the prefix. For any long edge  $(u, v)$ , let  $X_w$  denote the event that  $(u, w), (w, v)$  are both in the prefix. Note that the variables  $\{X_w\}_{w:r_u+1 \leq r_w \leq r_v-1}$  are negatively correlated and that

$$\Pr[X_w = 1] \geq (qT/M)^2/2 \geq \hat{q}^2/2.$$

Hence, if  $X = \sum_{w:r_u+1 \leq r_w \leq r_v-1} X_w$  then

$$\mathbb{E}[X] \geq c\hat{q}^{-2} \log n \cdot \hat{q}^2/2 = c/2 \cdot \log n$$

and so, by the Chernoff bound,  $X > 0$  with high probability and if this is the case, even if  $(u, v)$  is in the prefix, it will not be in the transitive reduction of the prefix. Hence, by the union bound, with high probability no long edges exist in the transitive closure of the prefix.  $\square$

### 2.1.8. Rank Aggregation

Recall the RANK-AGGR problem and the distance  $d$  between permutations, defined in Section 2.1.3. To recap, the distance between two orderings is the number of pairs of objects which are ranked differently by them, i.e.,

$$d(\pi, \sigma) := \sum_{a, b \in [n]} \mathbb{1}\{\pi(a) < \pi(b), \sigma(b) < \sigma(a)\}.$$

Note that RANK-AGGR is equivalent to finding the median of a set of  $k$  points under this distance function, which can be shown to be metric. It follows that picking a random ordering from the  $k$  input orderings provides a 2-approximation for RANK-AGGR.

A different approach is to reduce RANK-AGGR to the *weighted* feedback arc set problem on a tournament. This idea leads to a  $(1 + \varepsilon)$ -approximation via  $\ell_1$ -norm estimation in a way similar to the algorithm in Section 2.1.6. Define a vector  $\mathbf{x}$  of length  $\binom{n}{2}$  indexed by pairs of vertices  $\{a, b\}$  where

$$x_{a,b} = \sum_{i=1}^k \mathbb{1}\{\sigma_i(a) < \sigma_i(b)\},$$

i.e., the number of input orderings that have  $a < b$ . Then for any ordering  $\pi$  define a vector  $\mathbf{y}^\pi$ , where for each pair of vertices  $\{a, b\}$ ,

$$y_{a,b}^\pi = k \cdot \mathbb{1}\{\pi(a) < \pi(b)\}.$$

It is easy to see that  $\|\mathbf{x} - \mathbf{y}^\pi\|_1 = \text{cost}(\pi)$ .

As in Section 2.1.6, our algorithm maintains an  $\ell_1$ -sketch  $S\mathbf{x}$  with accuracy  $\varepsilon/3$  and error  $\delta = 1/(3 \cdot n!)$ . By Fact 2.1.1, this requires at most  $O(\varepsilon^{-2} n \log^2 n)$  space. In post-processing, the algorithm considers all  $n!$  permutations  $\pi$  and, for each of them, computes  $S(\mathbf{x} - \mathbf{y}^\pi) = S\mathbf{x} - S\mathbf{y}^\pi$ . It thereby recovers an estimate for  $\|\mathbf{x} - \mathbf{y}^\pi\|_1$  and finally outputs

the ordering  $\pi$  that minimizes this estimate.

The analysis of this algorithm is essentially the same as in Theorem 2.1.19. Overall, we obtain the following result.

**Theorem 2.1.35.** *There is a one-pass algorithm for rank aggregation that uses  $O(\varepsilon^{-2}n \log^2 n)$  space, returns a  $(1 + \varepsilon)$ -approximation with probability at least  $\frac{2}{3}$ , but requires exponential post-processing time.*  $\square$

### 2.1.9. Subsequent Works

Subsequent to our work, a number of works studied the multipass-streaming complexity of the  $s$ – $t$  reachability problem. They showed stronger lower bounds that also apply for the related problems of ACYC, TOPO-SORT, and FAS. Assadi and Raz [20] showed that any two-pass streaming algorithm for  $s$ – $t$  reachability on adversarial-order streams requires almost linear (in the number of edges), i.e.,  $\Omega(n^{2-o(1)})$  space, which significantly improves upon the  $\Omega(n^{7/6})$ -space lower bound of Guruswami and Onak. Chen et al. [66] then further improved the lower bound to show that the requirement of  $\Omega(n^{2-o(1)})$  space holds for any  $o(\sqrt{\log n})$ -pass algorithm.

Baweja, Jia, and Woodruff [35] improved upon our 3-approximation for FAS to give a polynomial-time  $(1 + \varepsilon)$ -approximation using  $p$  passes and  $O(n^{1+1/p})$  space. They also considered the problems of checking whether an input digraph is strongly connected (SCC) and finding the strongly connected components of a digraph (SCC-FIND). For each problem, they designed a  $(p + 1)$ -pass algorithm using  $O(n^{1+1/p})$ -space. By reduction from the SCI problem, they showed lower bounds for the SCC and SCC-FIND problems with similar tradeoffs as our lower bounds for reachability and other problems, i.e., SCC and SCC-FIND require  $n^{1+\Omega(1/p)}/p^{O(1)}$  space for  $p$  passes. Their work also provides some single-pass lower bound results: for digraphs on  $m$  edges, the problems SCC, ACYC, and determining whether there exists a path from a fixed vertex  $s$  to every other vertex in the graph, all require  $\Omega(m \log(n^2/m))$  space.

## Section 2.2

**Graph Coloring**

We now turn to the problem of vertex-coloring in the classical streaming model. Unlike the problems considered in the last section, this one is on *undirected* graphs. As we shall see, however, the concept of vertex orderings studied in the last section will play an important role in the design and analysis of our algorithms here as well.

Graph coloring is a fundamental topic in combinatorics and the corresponding algorithmic problem of coloring an input graph with few colors is a basic and heavily studied problem in computer science. It has numerous applications including in scheduling [126, 132, 165], air traffic flow management [33], frequency assignment in wireless networks [24, 149], and register allocation [52, 53, 68]. More recently, vertex coloring has been used to compute seed vertices in social networks that are then expanded to detect community structures in the network [143].

Given an  $n$ -vertex graph  $G = (V, E)$ , the task is to assign colors to the vertices in  $V$  so that no two vertices that share an edge get the same color. Doing so with the minimum possible number of colors—called the chromatic number,  $\chi(G)$ —is famously hard: it is NP-hard to even approximate  $\chi(G)$  to a factor of  $n^{1-\varepsilon}$  for any constant  $\varepsilon > 0$  [80, 120, 173]. In the face of this hardness, it is algorithmically interesting to color  $G$  with a possibly suboptimal number of colors depending upon tractable parameters of  $G$ . One such simple parameter is  $\Delta$ , the maximum degree. A trivial greedy algorithm colors  $G$  with  $\Delta + 1$  colors: go over the nodes in some arbitrary order and assign to each node the first color in  $[\Delta + 1]$  that is not already assigned to any of its neighbors (there is always an available color since a node has at most  $\Delta$  neighbors). This algorithm, however, uses linear space and time, which is bad news for massive real-world graphs. Thus, coloring with “about  $\Delta$ ” colors is a fairly non-trivial problem in sublinear settings such as streaming and *graph query* (also

called *sublinear time*). Further, the greedy algorithm is inherently *sequential*. In *parallel* or *distributed* computing models, it is a challenging and one of the most extensively studied problems (see for a more detailed discussion).

In a joint work with S.K. Bera [44], we initiated the study of graph coloring in the streaming model and obtained a  $\Delta(1 + o(1))$ -coloring algorithm in semi-streaming space. A parallel breakthrough work (awarded Best Paper at SODA 2019) by Assadi, Chen, and Khanna [17] then gave a tight  $(\Delta + 1)$ -coloring semi-streaming algorithm. Such a coloring, however, might sometimes use excessively larger number of colors than the optimal: think of star graph on  $n$  nodes which is 2-colorable but a  $(\Delta + 1)$ -coloring might use as many as  $n$  colors. This is the case for most *sparse* graphs in particular. Can we do better for such graphs?

Here, we focus on colorings that use “about  $\kappa$ ” colors, where  $\kappa = \kappa(G)$  is the *degeneracy* of  $G$ , a parameter that improves upon  $\Delta$ . It is defined as follows:  $\kappa = \min\{k : \text{every induced subgraph of } G \text{ has a vertex of degree at most } k\}$ . By definition,  $\kappa \leq \Delta$ . There is a simple greedy  $(\kappa + 1)$ -coloring algorithm analogous to the offline  $(\Delta + 1)$ -coloring algorithm mentioned above that runs using linear time and space; see Section 2.2.3. However, just as before, when processing a massive graph under the constraints of the space-bounded streaming model or the sublinear time and distributed computing models, the inherently sequential nature of the greedy algorithm makes it infeasible. We overcome this barrier with a very simple framework: decompose the graph into smaller subgraphs or blocks so as to store all the blocks in our limited memory, and then run the greedy algorithm on each block. We show that this basic framework suffices for obtaining a coloring with  $\kappa(1 + o(1))$  colors in semi-streaming space. Our analysis is simple, thanks to the concept of vertex orderings from the last section. Further, we show wide applicability of our framework: it can be easily implemented in the sublinear time and distributed computing models to obtain efficient colorings that improve upon the state of the art.

Degeneracy is closely related to the arboricity parameter  $\alpha$  of the graph, defined as the minimum number of forests into which the edges of  $G$  can be partitioned. Arboricity is more well-studied in the literature in the context of graph coloring: a long line of work has studied arboricity-based colorings in the dynamic and distributed models in order to save colors for sparse graphs. Every graph is  $2\alpha$ -colorable, and it is an easy exercise to show that  $\alpha \leq \kappa \leq 2\alpha - 1$ . Our  $\kappa(1 + o(1))$ -coloring not only provides a tighter bound on the number of colors, but also has simpler analysis than the corresponding  $2\alpha$  or  $O(\alpha)$ -coloring algorithms. Our work thus conveys an important conceptual message that degeneracy is a better parameter than arboricity in the context of graph coloring.

On the other hand, we give a number of lower bounds showing that, despite its simplicity, our algorithmic framework does about as good a job as *any* one-pass streaming algorithm can. In particular, no such algorithm can achieve  $(\kappa + O(1))$ -colorings without spending  $\Omega(n^2)$  space. Importantly, our lower bounds hold even if the value of the degeneracy  $\kappa$  of the graph is known to the algorithm in advance. At the same time, our upper bounds do not make any such assumptions.

### 2.2.1. Our Results and Techniques

**Streaming Algorithm.** We design a semi-streaming  $\kappa(1 + o(1))$ -coloring algorithm. More formally, we prove the following theorem.

**Theorem 2.2.1.** *(Short version of Theorem 2.2.8) There is a one-pass algorithm that processes a dynamic (i.e., insert-delete) graph stream using  $\tilde{O}(n)$  space and, with high probability, produces a  $\kappa(1 + o(1))$ -coloring. The post-processing at the end of the stream takes  $\tilde{O}(n)$  time.*

We briefly contrast this result with the previously known result of Assadi, Chen, and Khanna [17], which gives a  $(\Delta + 1)$ -coloring (see Section 2.2.2 for more detailed comparisons). As we have noted,  $\kappa \leq \Delta$  in every case; indeed,  $\kappa$  could be arbitrarily better than

$\Delta$  as shown by the example of a star graph, where  $\kappa = 1$  whereas  $\Delta = n - 1$ . From a practical standpoint, it is notable that in many real-world large graphs drawn from various application domains—such as social networks, web graphs, and biological networks—the parameter  $\kappa$  is often *significantly* smaller than  $\Delta$ . See Table 2.2 for some concrete numbers. Thus, our color bound is much better than [17] for a large class of graphs. That said,  $\kappa + o(\kappa)$  is mathematically incomparable with  $\Delta + 1$ .

| Graph Name        | $ V $ | $ E $ | $\Delta$ | $\kappa$ |
|-------------------|-------|-------|----------|----------|
| soc-friendster    | 66M   | 2B    | 5K       | 305      |
| fb-uci-uni        | 59M   | 92M   | 5K       | 17       |
| soc-livejournal   | 4M    | 28M   | 3K       | 214      |
| soc-orkut         | 3M    | 106M  | 27K      | 231      |
| web-baidu-baike   | 2M    | 18M   | 98K      | 83       |
| web-hudong        | 2M    | 15M   | 62K      | 529      |
| web-wikipedia2009 | 2M    | 5M    | 3K       | 67       |
| web-google        | 916K  | 5M    | 6K       | 65       |
| bio-mouse-gene    | 43K   | 14M   | 8K       | 1K       |
| bio-human-gene1   | 22K   | 12M   | 8K       | 2K       |
| bio-human-gene2   | 14K   | 9M    | 7K       | 2K       |
| bio-WormNet-v3    | 16K   | 763K  | 1K       | 165      |

Table 2.2: Statistics of several large real-world graphs taken from the application domains of social networks, web graphs, and biological networks, showing that the degeneracy,  $\kappa$ , is often significantly smaller than the maximum degree,  $\Delta$ . Source: <http://networkrepository.com> [154].

**Streaming Lower Bounds.** Recall that any graph with degeneracy  $\kappa$  has a proper  $(\kappa + 1)$ -coloring. Perhaps, analogous to Assadi et al.’s  $(\Delta + 1)$ -coloring algorithm [17], we could improve our algorithm’s color bound all the way to  $\kappa + 1$ ? We prove that this is not possible in sublinear space. In fact, our lower bounds prove more. We show that distinguishing  $n$ -vertex graphs of degeneracy  $\kappa$  from those with chromatic number  $\kappa + 2$  requires  $\Omega(n^2)$  space. This means that, in particular, it is hard to produce a  $(\kappa + 1)$ -coloring and to determine the exact value of  $\kappa$ . These results generalize to the problems of producing a  $(\kappa + \lambda)$ -coloring or estimating the degeneracy up to  $\pm\lambda$ ; the respective space lower

bounds generalize to  $\Omega(n^2/\lambda^2)$ . Furthermore, the streaming lower bounds hold even in the insertion-only model; compare this with our upper bound, which works even for dynamic graph streams.

A possible criticism of the above lower bounds for coloring is that they seem to depend on it being hard to *estimate* the degeneracy  $\kappa$ . Perhaps the coloring problem could become easier if  $\kappa$  was given to the algorithm in advance? We show another class of lower bounds establishing that this is not so: the same  $\Omega(n^2/\lambda^2)$  bound holds for any  $\lambda$  even with  $\kappa$  known *a priori*. Thus, specifically,  $(\kappa + 1)$ -coloring with prior knowledge of  $\kappa$  also requires  $\Omega(n^2)$  space. Recall that our algorithm, on the other hand, does not need to know  $\kappa$  in advance.

**Application to other space-conscious settings.** We apply the main framework used in our streaming algorithm to obtain coloring algorithms that achieve the same color bound in the following models: **(1)** the general graph query or sublinear time model [92], where we may access the graph using only neighbor queries (what is the  $i$ th neighbor of  $x$ ?) and pair queries (are  $x$  and  $y$  adjacent?); **(2)** the massively parallel communication (MPC) model, where each of a large number of memory-limited processors holds a sublinear-sized portion of the input data and computation proceeds using rounds of communication; **(3)** the congested-clique model of distributed computation, where there is one processor per vertex holding that vertex’s neighborhood information and each round allows each processor to communicate  $O(\log n)$  bits to a specific other processor; and **(4)** the LOCAL model of distributed computation, where there is one processor per vertex holding that vertex’s neighborhood information and each round allows each processor to send an arbitrary amount of information to all its neighbors.

Table 2.3 below summarizes our algorithmic results in each of these models and provides a basic comparison with the most relevant result from prior work; more detailed comparison appears in Section 2.2.2.

| Model     | Colors                     | Complexity Parameters  | Source    |
|-----------|----------------------------|--|-----------|
| Query     | $\Delta + 1$               | $\tilde{O}(n^{3/2})$ queries                                   | [17]      |
|           | $\kappa(1 + o(1))$         | $\tilde{O}(n^{3/2})$ queries                                   | this work |
| MPC       | $\Delta + 1$               | $O(1)$ rounds, $O(n \log^3 n)$ bits/proc                       | [17]      |
|           | $\kappa(1 + o(1))$         | $O(1)$ rounds, $O(n \log^2 n)$ bits/proc                       | this work |
| Congested | $\Delta + 1$               | $O(1)$ rounds  | [64]      |
| Clique    | $O(\kappa)$                | $O(1)$ rounds  | [88]      |
|           | $\kappa(1 + o(1))^*$       | $O(1)$ rounds  | this work |
| LOCAL     | $O(\alpha n^{1/k})$        | $O(k)$ rounds, $k \in [\omega(\log \log n), O(\sqrt{\log n})]$ | [124]     |
|           | $O(\alpha n^{1/k} \log n)$ | $O(k)$ rounds, $k \in [\omega(\sqrt{\log n}), o(\log n)]$      | this work |

Table 2.3: Summary of our algorithmic results and basic comparison with most relevant previous works. In the result marked (\*), we require that  $\kappa = \omega(\log^2 n)$ .

We also establish lower bounds in the query model analogous to the streaming setting: a  $(\kappa + 1)$ -coloring query algorithm needs  $\Omega(n^2)$  queries. More generally, distinguishing a graph with degeneracy  $\kappa$  from one with chromatic number  $\kappa + \lambda + 1$  requires  $\Omega(n^2/\lambda^2)$  queries, which means that estimating  $\kappa$  within an additive factor of  $\lambda$  or achieving a  $(\kappa + \lambda)$ -coloring requires  $\Omega(n^2/\lambda^2)$  queries for any  $\lambda$ . Also similar to the streaming model, we show that the coloring lower bound holds even if  $\kappa$  is known to the algorithm in advance.

**Techniques.** Perhaps even more than these results, our key contribution is a conceptual idea and a corresponding technical lemma underlying all our algorithms. We show that every graph admits a “small” sized *low degeneracy partition* (LDP), which is a partition of its vertex set into “few” blocks such that the subgraph induced by each block has low degeneracy, roughly logarithmic in  $n$ . Moreover, such an LDP can be computed by a very simple and distributed randomized algorithm: for each vertex, choose a “color” independently and uniformly at random from a suitable-sized palette (this is not to be confused

with the eventual graph coloring we seek; this random assignment is most probably not a proper coloring of the graph). The resulting color classes define the blocks of such a partition, with high probability. Theorem 2.2.6, the LDP Theorem, makes this precise. The proof of this theorem heavily uses vertex ordering arguments, a theme that we explored in Section 2.1.

Given an LDP, a generic graph coloring algorithm is to run a well-known offline  $(\kappa+1)$ -coloring greedy algorithm on each block, using distinct palettes for the distinct blocks. The fact that the resultant coloring is proper follows immediately. We then use the LDP Theorem to bound the number of colors and the space usage. We obtain algorithms achieving our claimed results in streaming as well as in the other models mentioned above by suitably implementing this generic algorithm in each computational model.

Our streaming lower bounds exploit the standard tool of communication complexity: for most of them, we use reductions from the INDEX problem via a novel gadget that we develop here; one bound uses a reduction from a variant of DISJ. These communication complexity problems are described in Section 1.3. Our query lower bounds use a related gadget and reductions from basic problems in *Boolean decision tree* complexity.

**A combinatorial lower bound.** A potential criticism of our algorithmic technique LDP is that it is rather simple; perhaps a more sophisticated graph-theoretic result, such as the Palette Sparsification Theorem (see below) of Assadi et al. [17], could improve the quality of the colorings obtained? In Section 2.2.8, we prove that this is not so: there is no analogous theorem for colorings with “about  $\kappa$ ” colors.

### 2.2.2. Related Work and Comparisons

---

**Streaming Model.** As mentioned before, our joint work with S.K. Bera [44] (on which our algorithm here builds) initiated the study of graph coloring in the streaming model and gave a one-pass semi-streaming  $\Delta(1 + o(1))$ -coloring algorithm. Assadi, Chen, and

Khanna [17] parallelly and independently gave a  $(\Delta + 1)$ -coloring algorithm using the same space and number of passes. Their algorithm exploits a key structural result that they establish: choosing a random  $O(\log n)$ -sized list from  $\{1, \dots, \Delta + 1\}$  for each vertex allows a compatible list coloring, i.e., a proper coloring where each node gets a color from its own list. They call this the *Palette Sparsification Theorem*. For our degeneracy-based coloring, while we do not get a similarly tight combinatorial result—none exists, as noted above—we do achieve faster post-processing time ( $\tilde{O}(n)$  versus their  $\tilde{O}(n\sqrt{\Delta})$ ). This win comes at the price of a less tight result— $(1 + o(1))\kappa$  colors instead of the combinatorially optimal  $\kappa + 1$ —but of course our lower bounds show that such slack is necessary. Also, as noted before, we often have  $\kappa \ll \Delta$  (Table 2.2). Further, for graphs with arboricity  $\alpha$  (see discussion at the beginning of Section 2.2 for definition and details), there was no previously known algorithm for  $O(\alpha)$ -coloring in the semi-streaming setting, whereas here we obtain a  $\kappa(1 + o(1))$ -coloring; recall the bound  $\kappa \leq 2\alpha - 1$ .

On the lower bound side, Abboud et al. [1] show that coloring a graph  $G$  with  $\chi(G)$  colors requires  $\Omega(n^2/p)$  space in  $p$  passes. They also show that deciding  $c$ -colorability for  $3 \leq c < n$  (that might be a function of  $n$ ) takes  $\Omega((n - c)^2/p)$  space in  $p$  passes. Furthermore, any streaming algorithm that distinguishes between  $\chi(G) \leq 3c$  and  $\chi(G) \geq 4c$  must use  $\Omega(n^2/p c^2)$  space. Another work on coloring in the streaming model prior to our work is the study of 2-coloring an  $n$ -uniform hypergraph by Radhakrishnan et al. [152]. Subsequent to our work, quite a few papers studied streaming graph coloring from multiple angles; see Section 2.2.9 for a discussion.

**Query Model.** Assadi et al. [17] also consider the graph coloring problem in the query model. They give a  $(\Delta + 1)$ -coloring algorithm that makes  $\tilde{O}(n^{3/2})$  queries, followed by a fairly elaborate computation that runs in  $\tilde{O}(n^{3/2})$  time and space. While our algorithm has the same space bound and number of queries, it is arguably much simpler: its post-processing is just the straightforward greedy offline algorithm for  $(\kappa + 1)$ -coloring.

Again, this simplification is probably possible because our final coloring is a  $(1 + o(1))$ -approximation to the combinatorially optimal  $(\kappa + 1)$  colors, whereas their bound is the optimal  $\Delta + 1$ . However, we do show that the super-constant slack for degeneracy-based coloring is a necessity (as opposed to degree-based coloring) in the query model as well. Also, Assadi et al. [17] proved a lower bound showing that any  $O(\Delta)$ -coloring requires  $\tilde{\Omega}(n^{3/2})$  queries, which also implies that our query-bound is tight.

**MPC and Congested Clique Models.** The MapReduce framework [74] is extensively used in distributed computing to process massive data sets. Beame, Koutris, and Succi [36] defined the Massively Parallel Communication (MPC)<sup>6</sup> model to abstract out key theoretical features of MapReduce; it has since become a widely used setting for designing and analyzing big data algorithms, especially for graph problems.

Another well studied model for distributed graph algorithms is Congested Clique [133]. Behnezhad et al. [38] show that Congested Clique is equivalent to the “semi-MPC model,” defined as MPC with  $O(n \log n)$  bits of memory per machine, thanks to simulations in both directions preserving the round complexity.

Harvey et al. [99] gave a  $(\Delta + o(\Delta))$ -coloring algorithm in the MapReduce model; it can be simulated in MPC using  $O(1)$  rounds and  $O(n^{1+c})$  space per machine for some constant  $c > 0$ . The aforementioned paper of Assadi et al. [17] gives an  $O(1)$ -round MPC algorithm for  $(\Delta + 1)$ -coloring using  $O(n \log^3 n)$  bits of space per machine. Because this space usage is  $\omega(n \log n)$ , the equivalence result of Behnezhad et al. [38] does not apply and this doesn’t lead to an  $O(1)$ -round Congested Clique algorithm. In contrast, our MPC algorithm can be made to use only  $O(n \log n)$  bits per machine and  $\kappa(1 + o(1))$  colors for graphs with  $\kappa = \omega(\log^2 n)$ , and therefore leads to such a Congested Clique algorithm. Chang et al. [64] gave an  $O(\sqrt{\log \log n})$ -round MPC algorithm with  $o(n)$  space per machine and  $\tilde{O}(m)$  space in total. Using the improved network decomposition results

<sup>6</sup>also known as Massively Parallel Computations

by Rozhon and Ghaffari [155], this round complexity can be reduced to  $O(\log \log \log n)$ . We, however, focus on the regime of quasi-linear memory per machine.

Graph coloring has recently garnered considerable attention in the Congested Clique model. Parter [150] gave a  $(\Delta + 1)$ -coloring algorithm using  $O(\log \log \Delta \cdot \log^* \Delta)$  rounds, later improved to  $O(\log^* \Delta)$  by Parter and Su [151]. Chang et al. [64] have improved this to  $O(1)$  rounds. They use similar but more involved graph partitioning techniques than us, as is probably necessary for a stringent  $(\Delta + 1)$ -coloring. For low-degeneracy graphs, our algorithm uses fewer colors than all these algorithms while achieving the best possible asymptotic round complexity ( $O(1)$ ). Parallel to our work, Ghaffari and Sayyadi [88] gave an  $O(1)$ -round algorithm for the  $O(\alpha)$ -coloring problem. Their analysis suggests that they obtain a  $(c\alpha)$ -coloring algorithm, where the constant  $c > 10$ . On the other hand, we get a tighter  $\kappa(1 + o(1))$ -coloring. Recall, again, that  $\kappa \leq 2\alpha - 1$  (Fact 2.2.2). Hence, we have an arguably simpler algorithmic framework achieving better results. The main novelty in our techniques lies in choosing degeneracy as the key parameter (instead of arboricity, which could lead to results looser by a factor of 2) and in the careful analysis that gives very sharp—not just asymptotic—bounds on the number of colors. Our algorithm (only the Congested Clique implementation), however, needs  $\kappa = \omega(\log^2 n)$  or  $\kappa = O(1)$  to keep the round complexity constant.

**The LOCAL Model.** The LOCAL model of distributed computing is “orthogonal” to Congested Clique: the input setup is similar but, during computation, each node may only communicate with its neighbors in the input graph, though it may send an arbitrarily long message. As before, the focus is on minimizing the number of rounds (a.k.a. time). There is a deep body of work on graph coloring in this model. Indeed, graph coloring is one of *the* most central “symmetry breaking” problems in distributed computing. We refer the reader to the monograph by Barenboim and Elkin [31] for an excellent overview of the state of the art. Here, we shall briefly discuss only a few results closely related to our contribution.

There is a long line of work on fast  $(\Delta + 1)$ -coloring in the LOCAL model, in the deterministic as well as the randomized setting [9, 28, 32, 84, 108, 134, 147, 157] culminating in sublogarithmic time solutions due to Harris [98] and Chang et al. [65]. Barenboim and Elkin [29, 30] studied fast distributed coloring algorithms that may use far fewer than  $\Delta$  colors: in particular, they gave algorithms that use  $O(\alpha)$  colors and run in  $O(\alpha^\varepsilon \log n)$  time on graphs with arboricity at most  $\alpha$ . Recall again that  $\kappa \leq 2\alpha - 1$ , so that a  $2\alpha$ -coloring always *exists*. They also gave a faster  $O(\log n)$ -time algorithm using  $O(\alpha^2)$  colors. Further, they gave a family of algorithms that produce an  $O(t\alpha^2)$ -coloring in  $O(\log_t n + \log^* n)$ , for every  $t$  such that  $2 \leq t \leq O(\sqrt{n/\alpha})$ . Our algorithm for the LOCAL model builds on this latter result.

Kothapalli and Pemmaraju [124] focused on arboricity-dependant coloring using very few rounds. They gave a randomized  $O(k)$ -round algorithm that uses  $O(\alpha n^{1/k})$  colors for  $2 \log \log n \leq k \leq \sqrt{\log n}$  and  $O(\alpha^{1+1/k} n^{1/k+3/k^2} 2^{-2^k})$  colors for  $k < 2 \log \log n$ . We extend their result to the range  $k \in [\omega(\sqrt{\log n}), o(\log n)]$ , using  $O(\alpha n^{1/k} \log n)$  colors.

Ghaffari and Lymouri [87] gave a randomized  $O(\alpha)$ -coloring algorithm that runs in time  $O(\log n \cdot \min\{\log \log n, \log \alpha\})$  as well as an  $O(\log n)$ -time algorithm using  $\min\{(2 + \varepsilon)\alpha + O(\log n \log \log n), O(\alpha \log \alpha)\}$  colors, for any constant  $\varepsilon > 0$ . However, their technique does not yield a sublogarithmic time algorithm, even at the cost of a larger palette.

**The LDP Technique.** As mentioned earlier, our algorithmic results rely on the concept of a low degeneracy partition (LDP) that we introduce in this work. Some relatives of this idea have been considered before. Specifically, Barenboim and Elkin [31] define a  $d$ -defective (resp.  $b$ -arbdefective)  $c$ -coloring to be a vertex coloring using palette  $[c]$  such that every color class induces a subgraph with maximum degree at most  $d$  (resp. arboricity at most  $b$ ). Obtaining such improper colorings is a useful first step towards obtaining proper colorings. They give deterministic algorithms to obtain good arbdefective colorings [30]. However, their algorithms are elaborate and are based on construction of low outdegree acyclic partial

orientations of the graph’s edges: an expensive step in our space-conscious models.

Elsewhere (Theorem 10.5 of Barenboim and Elkin [31]), they note that a useful defective (not arbdefective) coloring is easily obtained by randomly picking a color for each vertex; this is then useful for computing an  $O(\Delta)$ -coloring.

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

**The Degeneracy Parameter.** The parameter has been studied under several other names, such as *width* [85], *linkage* [121] and *Szekeres-Wilf number* [161]. For a graph  $G$ , the number  $\kappa(G) + 1$  is often called the *coloring number* of  $G$  [76, 160]. It has also been extensively studied as *k-core number* in different areas such as data streaming and parallel computing [77], distributed systems [12], data mining [142], protein networks [22], and social networks [48]. Farach-Colton and Tsai [79] studied the parameter in the streaming model, and gave a one-pass semi-streaming algorithm that approximates the degeneracy of an input graph within a multiplicative factor of  $1 + \varepsilon$ . Our lower bounds complement this result as we show that computing the degeneracy  $\kappa$  exactly or more generally within a multiplicative factor of  $(1 + \kappa^{-(1/2+\gamma)})$ , for some constant  $\gamma$ , is not possible in the one-pass semi-streaming setting.

**Other Related Work.** Other work considers coloring in the setting of *dynamic graph algorithms*: edges are inserted and deleted over time and the goal is to *maintain* a valid vertex coloring of the graph that must be updated quickly after each modification. Unlike in the streaming setting, there is no space restriction. Bhattacharya et al. [46] gave a randomized algorithm that maintains a  $(\Delta + 1)$ -coloring with  $O(\log \Delta)$  expected amortized update time and a deterministic algorithm that maintains a  $(\Delta + o(\Delta))$ -coloring with  $O(\text{polylog } \Delta)$  amortized update time. Barba et al. [27] gave tradeoffs between the number of colors used and update time. However, the techniques in these works do not seem to

apply in the streaming setting due to fundamental differences in the models.

Estimating the arboricity of a graph in the streaming model is a well studied problem. McGregor et al. [138] gave a one pass  $(1 + \varepsilon)$ -approximation algorithm to estimate the arboricity of graph using  $\tilde{O}(n)$  space. Bahmani et al. [23] gave a matching lower bound. Our lower bounds for estimating degeneracy are quantitatively much larger but they call for much tighter estimates.

### 2.2.3. Preliminary tools

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

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

**Fact 2.2.1.** *If an  $n$ -vertex graph has degeneracy  $\kappa$ , then it has at most  $\kappa n$  edges.* □

**Fact 2.2.2.** *In every graph, the degeneracy  $\kappa$  and arboricity  $\alpha$  satisfy  $\alpha \leq \kappa \leq 2\alpha - 1$ .* □

In analyzing our algorithms, it will be useful to consider certain *vertex orderings* of graphs and their connection with the notion of degeneracy, given by Lemma 2.2.4 below. Although the lemma is folklore, it is crucial to our analysis, so we include a proof for completeness.

**Definition 2.2.2.** An *ordering* of  $G$  is a list consisting of all its vertices (equivalently, a total order on  $V(G)$ ). Given an ordering  $\triangleleft$ , for each  $v \in V(G)$ , the *ordered neighborhood*

$$N_{G,\triangleleft}(v) := \{w \in V(G) : \{v, w\} \in E(G), v \triangleleft w\},$$

$$N_{G,\triangleleft}(v) := \{w \in V(G) : \{v, w\} \in E(G), v \triangleleft w\},$$

i.e., the set of neighbors of  $v$  that appear *after*  $v$  in the ordering. The *ordered degree*  $\text{odeg}_{G,\triangleleft}(v) := |N_{G,\triangleleft}(v)|$ .

**Definition 2.2.3.** A *degeneracy ordering* of  $G$  is an ordering produced by the following algorithm: starting with an empty list, repeatedly pick a minimum degree vertex  $v$  (breaking ties arbitrarily), append  $v$  to the end of the list, and delete  $v$  from  $G$ ; continue this until  $G$  becomes empty.

**Lemma 2.2.4.** A graph  $G$  is  $k$ -degenerate iff there exists an ordering  $\triangleleft$  such that  $\text{odeg}_{G,\triangleleft}(v) \leq k$  for all  $v \in V(G)$ .

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

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

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

**Lemma 2.2.5.** *Given unrestricted (“offline”) access to an input graph  $G$ , we can produce a proper  $(\kappa + 1)$ -coloring in linear time.*

*Proof.* Construct a degeneracy ordering  $(v_1, \dots, v_n)$  of  $G$  and then consider the vertices one by one in the order  $(v_n, \dots, v_1)$ , coloring greedily. Given a palette of size  $\kappa + 1$ , by the “only if” direction of Lemma 2.2.4, there will always be a free color for a vertex when it is considered.  $\square$

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

#### 2.2.4. LDP: A Generic Framework for Coloring

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

As a broad outline, our framework calls for coloring  $G$  in two phases. The first phase produces a *low degeneracy partition* (LDP) of  $G$ : it partitions  $V(G)$  into a “small” number of parts, each of which induces a subgraph that has “low” degeneracy. This step can be thought of as preprocessing and it is essentially free (in terms of complexity) in each of our models. The second phase properly colors each part, using a small number of colors, which is possible because the degeneracy is low. In Section 2.2.5, we shall see that the low degeneracy allows this second phase to be efficient in each of the models we consider.

**A Low Degeneracy Partition and its Application.** In this phase of our coloring framework, we assign each vertex a color chosen uniformly at random from  $[\ell]$ , these choices being

mutually independent, where  $\ell$  is a suitable parameter. For each  $i \in [\ell]$ , let  $G_i$  denote the subgraph of  $G$  induced by vertices colored  $i$ . We shall call each  $G_i$  a *block* of the vertex partition given by  $(G_1, \dots, G_\ell)$ . The next theorem, our main technical tool, provides certain guarantees on this partition given a suitable choice of  $\ell$ .

**Theorem 2.2.6** (LDP Theorem). *Let  $G$  be an  $n$ -vertex graph with degeneracy  $\kappa$ . Let  $k \in [1, n]$  be a “guess” for the value of  $\kappa$  and let  $s \geq Cn \log n$  be a sparsity parameter, where  $C$  is a sufficiently large universal constant. Put*

$$\ell = \left\lceil \frac{2nk}{s} \right\rceil, \quad \lambda = 3\sqrt{\kappa \ell \log n}, \quad (2.6)$$

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

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

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

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

**Definition 2.2.7.** Suppose graph  $G$  has degeneracy  $\kappa$ . A vertex partition  $(G_1, \dots, G_\ell)$  simultaneously satisfying the degeneracy bound in item (i), the block size bound in item (ii), and the (monochromatic) edge sparsity bound in item (iii) in Theorem 2.2.6 is called an  $(\ell, s, \lambda)$ -LDP of  $G$ .

It will turn out that an  $(\ell, s, \lambda)$ -LDP leads to a proper coloring of  $G$  using at most  $\kappa + \lambda + \ell$  colors. An instructive setting of parameters is  $s = \Theta((n \log n)/\varepsilon^2)$ , where  $\varepsilon$

is either a small constant or a slowly vanishing function of  $n$ , such as  $1/\log n$ . Then, a quick calculation shows that when an accurate guess  $k \in [\kappa, 2\kappa]$  is made, Theorem 2.2.6 guarantees an LDP that has edge sparsity  $s = \tilde{O}(n)$  and that leads to an eventual proper coloring using  $(1 + O(\varepsilon))\kappa$  colors. When  $\varepsilon = o(1)$ , this number of colors is  $\kappa + o(\kappa)$ .

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

$$\sum_{i=1}^{\ell} (\kappa(G_i) + 1) \leq \ell \left( \frac{\kappa + \lambda}{\ell} + 1 \right) = \kappa + \lambda + \ell, \quad (2.7)$$

as claimed above. Of course, even if our first phase random coloring  $\psi$  yields a suitable LDP, we still have to collect each block  $G_i$  or at least enough information about each block so as to produce a proper  $(\kappa(G_i) + 1)$ -coloring. How we do this depends on the precise model of computation; see Section 2.2.5 and Section 2.2.7.

**Proof of the LDP Theorem.** We now turn to proving the LDP Theorem from Section 2.2.4. Notice that when  $k \leq (C/2) \log n$ , the condition  $s \geq Cn \log n$  results in  $\ell = 1$ , so the vertex partition is the trivial one-block partition, which obviously satisfies all the properties in the theorem. Thus, in our proof, we may assume that  $k > (C/2) \log n$ .

*Proof of Theorem 2.2.6.* We start with item (ii), which is the most straightforward. From eq. (2.6), we have  $\ell \leq 4nk/s$ , so

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

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

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

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

Items (i) and (iii) include the condition  $k \leq 2\kappa$ , which we shall assume for the rest of the proof. By eq. (2.6) and the bounds  $s \geq Cn \log n$  and  $k > (C/2) \log n$ ,

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

whence, for sufficiently large  $C$ ,

$$\lambda \leq 3\sqrt{\kappa \cdot \frac{8\kappa}{C \log n} \cdot \log n} \leq \kappa. \quad (2.8)$$

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

By the “only if” direction of Lemma 2.2.4, we have  $\text{odeg}_{G, \triangleleft}(v) = |N_{G, \triangleleft}(v)| \leq \kappa$ .

Now note that

$$\text{odeg}_{G_j, \triangleleft_j}(v) = \sum_{u \in N_{G, \triangleleft}(v)} \mathbb{1}_{\{\psi(u) = \psi(v)\}}$$

is a sum of mutually independent indicator random variables, each of which has expectation  $1/\ell$ . Therefore,  $\mathbb{E} \text{odeg}_{G_j, \triangleleft_j}(v) = \text{odeg}_{G, \triangleleft}(v)/\ell \leq \kappa/\ell$ . Since  $\lambda \leq \kappa$  by eq. (2.8), we may use the form of the Chernoff bound in Fact 1.3.1, which gives us

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

where the equality follows from eq. (2.6). In words, with probability at least  $1 - 1/n^3$ , the vertex  $v$  has ordered degree at most  $(\kappa + \lambda)/\ell$  within its own block. By a union bound, with probability at least  $1 - 1/n^2$ , all  $n$  vertices of  $G$  satisfy this property. When this happens, by the “if” direction of Lemma 2.2.4, it follows that  $\kappa(G_i) \leq (\kappa + \lambda)/\ell$  for every  $i$ .

Finally, we take up item (iii), which is now straightforward. Assume that the high probability event in item (i) occurs. Then, by Fact 2.2.1,

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

where the final inequality uses the condition  $\kappa \leq k$  and eq. (2.6).  $\square$

### 2.2.5. Streaming Algorithm for Degeneracy-Based Coloring

For graph problems, in the basic streaming model, the input is a stream of non-repeated edges that define the input graph  $G$ : this is called the *insertion-only* model, since it can be thought of as building up  $G$  through a sequence of edge insertions. In the more general *dynamic graph model* or *turnstile model*, the stream is a sequence of edge updates, each update being either an insertion or a deletion: the net effect is to build up  $G$ . Our algorithm will work in this more general model. Later, we shall give a corresponding lower bound that will hold even in the insertion-only model (for a lower bound, this is a strength).

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

at the end. We shall say that  $\sigma$  *builds up*  $\mathbf{x}$  and  $G$ .

Our algorithm makes use of two data streaming primitives, each a *linear sketch*. (We can do away with these sketches in the insertion-only setting; see the end of this section.) The first is a sketch for *sparse recovery* given by a matrix  $A$  (say): given a vector  $\mathbf{x} \in [[m]]^N$  with sparsity  $\|\mathbf{x}\|_0 \leq t$ , there is an efficient algorithm to reconstruct  $\mathbf{x}$  from  $A\mathbf{x}$ . The second is a sketch for  $\ell_0$  *estimation* given by a random matrix  $B$  (say): given a vector  $\mathbf{x} \in [[m]]^N$ , there is an efficient algorithm that takes  $B\mathbf{x}$  and computes from it an estimate of  $\|\mathbf{x}\|_0$  that, with probability at least  $1 - \delta$ , is a  $(1 + \gamma)$ -multiplicative approximation. It is known that there exists a suitable  $A \in \{0, 1\}^{d \times N}$ , where  $d = O(t \log(N/t))$ , where  $A$  has column sparsity  $O(\log(N/t))$ ; see, e.g., Theorem 9 of Gilbert and Indyk [90]. It is also known that there exists a suitable distribution over matrices giving  $B \in \{0, 1\}^{d' \times N}$  with  $d' = O(\gamma^{-2} \log \delta^{-1} \log N (\log \gamma^{-1} + \log \log m))$ . Further, given an update to the  $i$ th entry of  $\mathbf{x}$ , the resulting updates in  $A\mathbf{x}$  and  $B\mathbf{x}$  can be effected quickly by generating the required portion of the  $i$ th columns of  $A$  and  $B$ .

---

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

---

```

1: procedure COLOR(stream  $\sigma$ , integer  $k$ )  $\triangleright \sigma$  builds up  $\mathbf{x}$  and  $G$ ;  $k \in [1, n]$  is a guess
   for  $\kappa(G)$ 
2:   choose  $s, \ell$  as in eq. (2.6) and  $t, d, d', A, B$  as in the above discussion
3:   initialize  $\mathbf{y} \in [[m]]^d$  and  $\mathbf{z} \in [[m]]^{d'}$  to zero
4:   foreach  $u \in [n]$  do  $\psi(u) \leftarrow$  uniform random color in  $[\ell]$ 
5:   foreach token  $(u, v, c)$  in  $\sigma$  do
6:     if  $\psi(u) = \psi(v)$  then  $\mathbf{y} \leftarrow \mathbf{y} + cA_{u,v}$ ;  $\mathbf{z} \leftarrow \mathbf{z} + cB_{u,v}$ 
7:     if estimate of  $\|\mathbf{w}\|_0$  obtained from  $\mathbf{z}$  is  $> 5s/4$  then abort
8:      $\mathbf{w}' \leftarrow$  result of  $t$ -sparse recovery from  $\mathbf{y}$   $\triangleright$  we expect that  $\mathbf{w}' = \mathbf{w}$ 
9:     foreach  $i \in [\ell]$  do
10:       $G_i \leftarrow$  simple graph induced by  $\{\{u, v\} : w'_{u,v} \neq 0 \text{ and } \psi(u) = \psi(v) = i\}$ 
11:      color  $G_i$  using palette  $\{(i, j) : 1 \leq j \leq \kappa(G_i) + 1\}$ ; cf. Lemma 2.2.5  $\triangleright$  net
effect is to color  $G$ 

```

---

In our description of Algorithm 1, we use  $A_{u,v}$  (resp.  $B_{u,v}$ ) to denote the column of  $A$  (resp.  $B$ ) indexed by  $\{u, v\}$ . The algorithm's logic results in sketches  $\mathbf{y} = A\mathbf{w}$  and

$\mathbf{z} = B\mathbf{w}$ , where  $\mathbf{w}$  corresponds to the subgraph of  $G$  consisting of  $\psi$ -monochromatic edges only (cf. Theorem 2.2.6), i.e.,  $\mathbf{w}$  is obtained from  $\mathbf{x}$  by zeroing out all entries except those indexed by  $\{u, v\}$  with  $\psi(u) = \psi(v)$ . We choose the parameter  $t = 2s$ , where  $s \geq Cn \log n$  is the sparsity parameter from Theorem 2.2.6, which gives  $d = O(s \log n)$ ; we choose  $\gamma = 1/4$  and  $\delta = 1/n$ , giving  $d' = O(\log^3 n)$ .

Notice that Algorithm 1 requires a guess for  $\kappa := \kappa(G)$ , which is not known in advance. Our final one-pass algorithm runs  $O(\log n)$  parallel instances of  $\text{COLOR}(\sigma, k)$ , using geometrically spaced guesses  $k = 2, 4, 8, \dots$ . It outputs the coloring produced by the non-aborting run that uses the smallest guess.

**Theorem 2.2.8.** *Set  $s = \lceil \varepsilon^{-2} n \log n \rceil$ , where  $\varepsilon > 0$  is a parameter. The above one-pass algorithm processes a dynamic (i.e., turnstile) graph stream using  $O(\varepsilon^{-2} n \log^4 n)$  bits of space and, with high probability, produces a proper coloring using at most  $(1 + O(\varepsilon))\kappa$  colors. In particular, taking  $\varepsilon = 1/\log n$ , it produces a  $\kappa + o(\kappa)$  coloring using  $\tilde{O}(n)$  space. Each edge update is processed in  $\tilde{O}(1)$  time and post-processing at the end of the stream takes  $\tilde{O}(n)$  time.*

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

We start with (a). Consider a particular run using  $k \leq k^*$ . By item (i) of Theorem 2.2.6, each  $G_i$  has degeneracy at most  $(\kappa + \lambda)/\ell$ ; so if  $\mathbf{w}$  is correctly recovered by the sparse recovery sketch (i.e.,  $\mathbf{w}' = \mathbf{w}$  in Algorithm 1), then each  $G_i$  is correctly recovered and the run uses at most  $\kappa + \lambda + \ell$  colors, as in eq. (2.7). Using the values from eq. (2.6), this number is at most  $(1 + O(\varepsilon))\kappa$ . Now, if the run does not abort, then the estimate of the sparsity  $\|\mathbf{w}\|_0$  is at most  $5s/4$ . By the guarantees of the  $\ell_0$ -estimation sketch, the true sparsity is at most  $(5/4)(5s/4) < 2s = t$ , so, w.h.p.,  $\mathbf{w}$  is indeed  $t$ -sparse and, by the guarantees of the

sparse recovery sketch,  $\mathbf{w}' = \mathbf{w}$ . Taking a union bound over all  $O(\log n)$  runs, the bound on the number of colors holds for all required runs simultaneously, w.h.p.

We now take up (b). Note that  $\|\mathbf{w}\|_0$  is precisely the number of  $\psi$ -monochromatic edges in  $G$ . By item (iii) of Theorem 2.2.6, we have  $\|\mathbf{w}_0\| \leq s$  w.h.p. By the accuracy guarantee of the  $\ell_0$ -estimation sketch, in this run the estimate of  $\|\mathbf{w}\|_0$  is at most  $5s/4$  w.h.p., so the run does not abort.

The space usage of each parallel run is dominated by the computation of  $\mathbf{y}$ , so it is  $O(d \log m) = O(s \log n \log m) = O(\varepsilon^{-2} n \log^3 n)$ , using our setting of  $s$  and the assumption  $m = \text{poly}(n)$ . The claims about the update time and post-processing time follow directly from the properties of a state-of-the-art sparse recovery scheme, e.g., the scheme based on expander matching pursuit given in Theorem 9 of Gilbert and Indyk [90].  $\square$

**Simplification for Insertion-Only Streams.** Algorithm 1 can be simplified considerably if the input stream is insertion-only. We can then initialize each  $G_i$  to an empty graph and, upon seeing an edge  $\{u, v\}$  in the stream, insert it into  $G_i$  iff  $\psi(u) = \psi(v) = i$ . We abort if we collect more than  $s$  edges; w.h.p., this will not happen, thanks to Theorem 2.2.6. Finally, we color the collected graphs  $G_i$  greedily, just as in Algorithm 1. With this simplification, the overall space usage drops to  $O(s \log n) = O(\varepsilon^{-2} n \log^2 n)$  bits.

The reason this does not work for dynamic graph streams is that the number of monochromatic edges could exceed  $s$  by an arbitrary amount mid-stream.

### 2.2.6. Streaming Lower Bounds

We investigate whether we can improve the number of colors used by our algorithms to  $\kappa + 1$ , rather than  $\kappa(1 + o(1))$ ? After all, every graph  $G$  does have a proper  $(\kappa(G) + 1)$ -coloring. The main message of this section is that answer is a strong “No”. If we insist on a coloring that good, we would incur the worst possible space complexity:  $\Omega(n^2)$ . In fact, it holds even if the input stream consists of edge insertions alone. Furthermore, this holds

even if  $\kappa$  is known to the algorithm in advance.

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

**Definition 2.2.9.** Let  $G$  be a graph and  $\lambda$  a positive integer. The *blow-up graph*  $G^\lambda$  is obtained by replacing each vertex of  $G$  with a copy of the complete graph  $K_\lambda$  and replacing each edge of  $G$  with a complete bipartite graph between the copies of  $K_\lambda$  at its endpoints. More succinctly,  $G^\lambda$  is the lexicographical product  $G[K_\lambda]$ .

**Lemma 2.2.10** (Blow-Up Lemma). *For all graphs  $G$  and positive integers  $\lambda, c$ , if  $G$  has a  $c$ -clique, then  $G^\lambda$  has a  $(c\lambda)$ -clique. Also,  $\kappa(G^\lambda) \leq (\kappa(G) + 1)\lambda - 1$ .*

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

Our lower bounds come in two flavors. The first address the hardness of distinguishing low-degeneracy graphs from high-chromatic-number graphs. This is encapsulated in the following abstract problem.

**Definition 2.2.11** (GRAPH-DIST problem). Consider two graph families:  $\mathcal{G}_1 := \mathcal{G}_1(n, q, \lambda)$ , consisting of  $n$ -vertex graphs with chromatic number  $\chi \geq (q + 1)\lambda$ , and  $\mathcal{G}_2 := \mathcal{G}_2(n, q, \lambda)$ , consisting of  $n$ -vertex graphs with  $\kappa \leq q\lambda - 1$ . Then  $\text{GRAPH-DIST}(n, q, \lambda)$  is the problem of distinguishing  $\mathcal{G}_1$  from  $\mathcal{G}_2$ ; note that  $\mathcal{G}_1 \cap \mathcal{G}_2 = \emptyset$ . More precisely, given an input graph  $G$  on  $n$  vertices, the problem is to decide whether  $G \in \mathcal{G}_1$  or  $G \in \mathcal{G}_2$ , with success probability at least  $2/3$ .

We shall prove that GRAPH-DIST is “hard” in the insertion-only streaming setting and in the query setting, thereby establishing that in these models it is hard to produce a  $(\kappa + \lambda)$ -coloring. In fact, our proofs will show that it is just as hard to estimate the parameter  $\kappa$ ;

this goes to show that the hardness of the coloring problem is not just because of the large output size.

Lower bounds of the above flavor raise the following question: since estimating  $\kappa$  itself is hard, does the coloring problem become easier if the value of  $\kappa(G)$  is given in advance, before the algorithm starts to read  $G$ ? In fact, the  $(\Delta + 1)$ -coloring algorithms by Assadi et al. [17] assume that  $\Delta$  is known in advance. However, perhaps surprisingly, we prove a second flavor of lower bounds, showing that *a priori* knowledge of  $\kappa$  does not help and  $(\kappa + 1)$ -coloring (more generally,  $(\kappa + \lambda)$ -coloring) remains a hard problem even under the strong assumption that  $\kappa$  is known in advance.

The above tools not only help in proving streaming lower bounds as we describe next, but also in proving lower bounds in the query model as demonstrated in Section 4.3.6.

We prove two flavors of lower bounds in the one-pass streaming setting. Our streaming lower bounds use reductions from the INDEX and INT-FIND (intersection finding, a variant of DISJOINTNESS) problems in communication complexity (see Section 1.3).

In INT-FIND<sub>N</sub>, Alice and Bob hold vectors  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^N$ , interpreted as subsets of  $[N]$ , satisfying the promise that  $|\mathbf{x} \cap \mathbf{y}| = 1$ . They must find the unique index  $i$  where  $x_i = y_i = 1$ , using at most  $c$  bits of randomized interactive communication, succeeding with probability at least  $2/3$ . The smallest  $c$  for which such a protocol exists is the randomized communication complexity,  $R(\text{INT-FIND}_N)$ . Recall that  $R^\rightarrow(\text{INDEX}_N) = \Omega(N)$  (Fact 1.3.2) and  $R(\text{INT-FIND}_N) = \Omega(N)$ ; the latter is a simple extension of the DISJOINTNESS lower bound (Fact 1.3.3).

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

**Definition 2.2.12.** For  $\mathbf{x} \in \{0, 1\}^{p \times p}$ , a realization of  $H_{\mathbf{x}}$  on a list  $(\ell_1, \dots, \ell_p, r_1, \dots, r_p)$  of distinct vertices is a graph on these vertices whose edge set is  $\{\{\ell_i, r_j\} : x_{ij} = 1\}$ .

**First Flavor: Degeneracy Not Known in Advance.** To prove lower bounds of the first flavor, we start by demonstrating the hardness of the abstract problem GRAPH-DIST, from Definition 2.2.11.

**Lemma 2.2.13.** *Solving GRAPH-DIST( $n, q, \lambda$ ) in one randomized streaming pass requires  $\Omega(n^2/\lambda^2)$  space.*

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

*Proof.* Put  $p = q - 1$ . We reduce from INDEX $_N$ , where  $N = p^2$ , using the following plan. Starting with an empty graph on  $n = 3\lambda p$  vertices, Alice adds certain edges based on her input  $\mathbf{x} \in \{0, 1\}^{p \times p}$  and then Bob adds certain other edges based on his input  $(y, z) \in [p] \times [p]$ . By design, solving GRAPH-DIST( $n, q, \lambda$ ) on the resulting final graph reveals the bit  $x_{yz}$ , implying that a one-pass streaming algorithm for GRAPH-DIST requires at least  $R^+(\text{INDEX}_N) = \Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$  bits of memory. The details follow.

We first consider  $\lambda = 1$ . We use the vertex set  $L \uplus R \uplus C$  (the notation “ $\uplus$ ” denotes a disjoint union), where  $L = \{\ell_1, \dots, \ell_p\}$ ,  $R = \{r_1, \dots, r_p\}$ , and  $|C| = p$ . Alice introduces the edges of the gadget graph  $H_{\mathbf{x}}$  (from Definition 2.2.12), realized on the vertices  $(\ell_1, \dots, \ell_p, r_1, \dots, r_p)$ . Bob introduces all possible edges within  $C \cup \{\ell_y, r_z\}$ , except for  $\{\ell_y, r_z\}$ . Let  $G$  be the resulting graph (see Figure 2.1).

If  $x_{yz} = 1$ , then  $G$  contains a clique on  $C \cup \{\ell_y, r_z\}$ , whence  $\chi(G) \geq p + 2$ . If, on the other hand,  $x_{yz} = 0$ , then we claim that  $\kappa(G) \leq p$ . By Lemma 2.2.4, the claim will follow if we exhibit a vertex ordering  $\triangleleft$  such that  $\text{odeg}_{G, \triangleleft}(v) \leq p$  for all  $v \in V(G)$ . We use an ordering where

$$L \cup R \setminus \{\ell_y, r_z\} \triangleleft \ell_y \triangleleft \{r_z\} \cup C$$

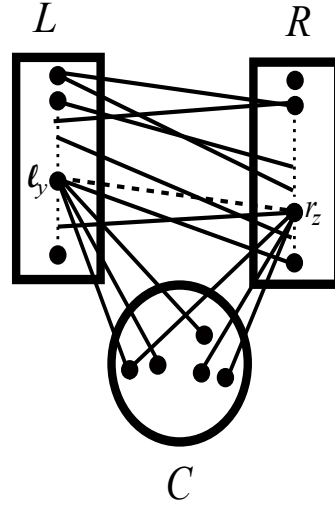


Figure 2.1: Gadget graph for proving lower bounds of first flavor

and the ordering within each set is arbitrary. By construction of  $H_x$ , each vertex in  $L \cup R \setminus \{\ell_y, r_z\}$  has *total* degree at most  $p$ . For each vertex  $v \in \{r_z\} \cup C$ , we trivially have  $\text{odeg}_{G, \triangleleft}(v) \leq p$  because  $|C| = p$ . Finally, since  $x_{yz} = 0$ , the vertex  $r_z$  is not a neighbor of  $\ell_y$ ; so  $\text{odeg}_{G, \triangleleft}(\ell_y) = |C| = p$ . This proves the claim.

When  $\lambda \geq 1$ , Alice and Bob introduce edges so as to create the blow-up graph  $G^\lambda$ , as in Definition 2.2.9. By Lemma 2.2.10, if  $x_{yz} = 1$ , then  $G^\lambda$  has a  $(p+2)\lambda$ -clique, whereas if  $x_{yz} = 0$ , then  $\kappa(G^\lambda) \leq (p+1)\lambda - 1$ . In the former case,  $\chi(G^\lambda) \geq (p+2)\lambda = (q+1)\lambda$ , so that  $G^\lambda \in \mathcal{G}_1(n, q, \lambda)$ ; cf. Definition 2.2.11. In the latter case,  $\kappa(G^\lambda) \leq q\lambda - 1$ , so that  $G^\lambda \in \mathcal{G}_2(n, q, \lambda)$ . Thus, solving  $\text{GRAPH-DIST}(n, q, \lambda)$  on  $G^\lambda$  reveals  $x_{yz}$ .  $\square$

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

**Theorem 2.2.14.** *Given a single randomized pass over a stream of edges of an  $n$ -vertex graph  $G$ , succeeding with probability at least  $2/3$  at either of the following tasks requires  $\Omega(n^2/\lambda^2)$  space, where  $\lambda \geq 1$  is an integer parameter:*

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

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

*Proof.* An algorithm for either task (i) and or task (ii) immediately solves GRAPH-DIST with appropriate parameters, implying the  $\Omega(n^2/\lambda^2)$  bounds, thanks to Lemma 2.2.13. For the “furthermore” statement, note that the graphs in the family  $\mathcal{G}_2$  constructed in the proof of Lemma 2.2.13 have  $\kappa = \Theta(n)$ , so performing either task with the stated guarantee on  $\lambda$  would require  $\Omega(n^{1+2\gamma})$  space, which is not in  $\tilde{O}(n)$ .  $\square$

Combining the above result with the algorithmic result in Theorem 2.2.8, we see that producing a  $\kappa(1 + o(1))$ -coloring is possible in semi-streaming space whereas producing a  $(\kappa + O(\kappa^{\frac{1}{2}-\gamma}))$ -coloring is not. We leave open the question of whether this gap can be tightened.

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

**Theorem 2.2.15.** *Given as input an integer  $\kappa$ , followed by a stream of edges of an  $n$ -vertex graph  $G$  with degeneracy  $\kappa$ , a randomized one-pass algorithm that produces a proper  $(\kappa + \lambda)$ -coloring of  $G$  requires  $\Omega(n^2/\lambda^2)$  bits of space. Furthermore, if we require  $\lambda = O(\kappa^{\frac{1}{2}-\gamma})$ , where  $\gamma > 0$ , then the task does not admit a semi-streaming algorithm.*

*Proof.* We reduce from INDEX<sub>N</sub>, where  $N = p^2$ , using a plan analogous to the one used in proving Lemma 2.2.13. Alice and Bob will construct a graph on  $n = 5\lambda p$  vertices, using their respective inputs  $\mathbf{x} \in \{0, 1\}^{p \times p}$  and  $(y, z) \in [p] \times [p]$ .

First, we consider the case  $\lambda = 1$ . We use the vertex set  $L \uplus R \uplus \bar{L} \uplus \bar{R} \uplus C$ , where  $L = \{\ell_1, \dots, \ell_p\}$ ,  $R = \{r_1, \dots, r_p\}$ ,  $\bar{L} = \{\bar{\ell}_1, \dots, \bar{\ell}_p\}$ ,  $\bar{R} = \{\bar{r}_1, \dots, \bar{r}_p\}$ , and  $|C| = p$ . Let  $\bar{\mathbf{x}}$  be the bitwise complement of  $\mathbf{x}$ . Alice introduces the edges of the gadget graph  $H_{\mathbf{x}}$  (from Definition 2.2.12), realized on  $L \cup R$ , and the edges of  $H_{\bar{\mathbf{x}}}$  realized on  $\bar{L} \cup \bar{R}$ . For

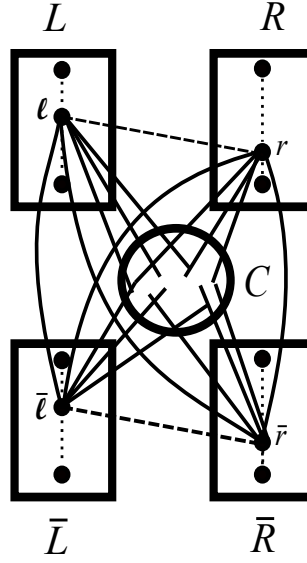


Figure 2.2: Gadget graph for proving lower bounds of second flavor

ease of notation, put  $\ell := \ell_y$ ,  $r := r_z$ ,  $\bar{\ell} := \bar{\ell}_y$ ,  $\bar{r} := \bar{r}_z$ , and  $S := C \cup \{\ell, r, \bar{\ell}, \bar{r}\}$ . Bob introduces all possible edges within  $S$ , except for  $\{\ell, r\}$  and  $\{\bar{\ell}, \bar{r}\}$ . Let  $G$  be the resulting graph (see Figure 2.2).

We claim that the degeneracy  $\kappa(G) = p+2$ . To prove this, we consider the case  $x_{yz} = 1$  (the other case,  $x_{yz} = 0$ , is symmetric). By construction,  $G$  contains a clique on the  $p+3$  vertices in  $C \cup \{\ell, r, \bar{\ell}\}$ ; therefore, by definition of degeneracy,  $\kappa(G) \geq p+2$ . To show that  $\kappa(G) \leq p+2$ , it will suffice to exhibit a vertex ordering  $\triangleleft$  such that  $\text{odeg}_{G, \triangleleft}(v) \leq p+2$  for all  $v \in V(G)$ . To this end, consider an ordering where

$$V(G) \setminus S \triangleleft \bar{\ell} \triangleleft S \setminus \{\bar{\ell}\}$$

and the ordering within each set is arbitrary. Each vertex  $v \in V(G) \setminus S$  has  $\text{odeg}_{G, \triangleleft}(v) \leq \deg(v) \leq p$  and each vertex  $v \in S \setminus \{\bar{\ell}\}$  has  $\text{odeg}_{G, \triangleleft}(v) \leq |S \setminus \{\bar{\ell}\}| - 1 = p+2$ . As for the vertex  $\bar{\ell}$ , since  $\bar{x}_{yz} = 1 - x_{yz} = 0$ , by the construction in Definition 2.2.12,  $\bar{r}$  is not a neighbor of  $\bar{\ell}$ ; therefore,  $\text{odeg}_{G, \triangleleft}(\bar{\ell}) \leq |S \setminus \{\bar{\ell}, \bar{r}\}| = p+2$ .

Let  $\mathcal{A}$  be a streaming algorithm that behaves as in the theorem statement. Recall that

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

Now consider an arbitrary  $\lambda$ . Alice and Bob proceed as above, except that they simulate  $\mathcal{A}$  on the blow-up graph  $G^\lambda$ . Since  $G$  always has a  $(p + 3)$ -clique and  $\kappa(G) = p + 2$ , the two halves of Lemma 2.2.10 together imply  $\kappa(G^\lambda) = (p + 3)\lambda - 1$ . So, when  $\mathcal{A}$  succeeds, it properly colors  $G^\lambda$  using at most  $(p + 4)\lambda - 1$  colors. For each  $A \subseteq V(G)$ , abusing notation, let  $A^\lambda$  denote its corresponding set of vertices in  $G^\lambda$  (cf. Definition 2.2.9). Since  $|S^\lambda| = (p + 4)\lambda$ , there must be a color repetition within  $S^\lambda$ . Reasoning as above, this repetition must occur within  $\{\ell, r\}^\lambda$  when  $x_{yz} = 0$  and within  $\{\bar{\ell}, \bar{r}\}^\lambda$  when  $x_{yz} = 1$ . Therefore, Bob can examine the coloring to solve  $\text{INDEX}_N$ , showing that  $\mathcal{A}$  must use  $\Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$  space.

The “furthermore” part follows by observing that  $\kappa(G^\lambda) = \Theta(|V(G^\lambda)|)$ . □

**Multiple Passes.** The streaming algorithm from Section 2.2.5 is one-pass, as are the lower bounds proved above. Is the coloring problem any easier if we are allowed multiple passes over the edge stream? We now give a simple argument showing that, if we slightly generalize the problem, it stays just as hard using multiple ( $O(1)$  many) passes.

The generalization is to allow some edges to be *repeated* in the stream. In other words, the input is a multigraph  $\hat{G}$ . Clearly, a coloring is proper for  $\hat{G}$  iff it is proper for the underlying simple graph  $G$ , so the relevant algorithmic problem is to properly  $(\kappa + \lambda)$ -color  $G$ , where  $\kappa := \kappa(G)$ . Note that our algorithm in Section 2.2.5 does, in fact, solve this

more general problem.

**Theorem 2.2.16.** *Given as input an integer  $\kappa$ , followed by a stream of edges of an  $n$ -vertex multigraph  $\hat{G}$  whose underlying simple graph has degeneracy  $\kappa$ , a randomized  $p$ -pass algorithm that produces a proper  $(\kappa + \lambda)$ -coloring of  $G$  requires  $\Omega(n^2/(\lambda^2 p))$  bits of space. This holds even if the stream is insertion-only, with each edge appearing at most twice.*

*Proof.* As usual, we prove this for  $\lambda = 1$  and appeal to the Blow-Up Lemma (Lemma 2.2.10) to generalize.

We reduce from INT-FIND $_N$ , with  $N = \binom{n}{2}$ . Let Alice and Bob treat their inputs as  $(x_{ij})_{1 \leq i < j \leq n}$  and  $(y_{ij})_{1 \leq i < j \leq n}$  in some canonical way. Alice (resp. Bob) converts their input into an edge stream consisting of pairs  $(i, j)$  such that  $i < j$  and  $x_{ij} = 0$  (resp.  $y_{ij} = 0$ ). The concatenation of these streams defines the multigraph  $\hat{G}$  given to the coloring algorithm. Let  $(h, k)$  be the unique pair such that  $x_{hk} = y_{hk} = 1$ . Note that the underlying simple graph  $G$  is  $K_n$  minus the edge  $\{h, k\}$ . Therefore,  $\kappa = n - 2$  and so, in a proper  $(n - 1)$ -coloring of  $\hat{G}$ , there must be a repeated color and this can only happen at vertices  $h$  and  $k$ .

Thus, a  $p$ -pass  $(\kappa + 1)$ -coloring algorithm using  $s$  bits of space leads to a protocol for INT-FIND $_N$  using  $(2p - 1)s$  bits of communication. Therefore,  $s = \Omega(N/p) = \Omega(n^2/p)$ . □

### 2.2.7. Applications in Various Space-Conscious Models

We now turn to designing graph coloring algorithms in models for big data computation other than streaming, all of which deal with the challenge posed by the size of a massive input graph. We call such models *space-conscious*. They include the general graph query model and certain distributed models of computation such as MPC, Congested Clique, and LOCAL. In each case, our algorithm ultimately relies on the framework developed in

Section 2.2.4. For the query model, we also give complementary lower bounds.

**Query Model.** The *general graph query model* is a standard model of space-conscious algorithms for big graphs where the input graph is random-accessible but the emphasis is on the examining only a tiny (ideally, sublinear) portion of it; for general background see Chapter 10 of Goldreich’s book [92]. In this model, the algorithm starts out knowing the vertex set  $[n]$  of the input graph  $G$  and can access  $G$  only through the following types of queries.

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

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

By adapting the combinatorial machinery from their semi streaming algorithm, Assadi et al. [17] gave an  $\tilde{O}(n^{3/2})$ -query algorithm for finding a  $(\Delta + 1)$ -coloring. Our LDP framework gives a considerably simpler algorithm using  $\kappa + o(\kappa)$  colors, where  $\kappa := \kappa(G)$ . We remark here that  $\tilde{O}(n^{3/2})$  query complexity is essentially optimal, as Assadi et al. [17] proved a matching lower bound for any  $(c \cdot \Delta)$ -coloring algorithm, for any constant  $c > 1$ .

**Theorem 2.2.17.** *Given query access to a graph  $G$ , there is a randomized algorithm that, with high probability, produces a proper coloring of  $G$  using  $\kappa + o(\kappa)$  colors. The algorithm’s worst-case query complexity, running time, and space usage are all  $\tilde{O}(n^{3/2})$ .*

*Proof.* The algorithm proceeds in two stages. In the first stage, it attempts to extract all edges in  $G$  through neighbor queries alone, aborting when “too many” queries have

been made. More precisely, it loops over all vertices  $v$  and, for each  $v$ , issues queries  $\text{Neighbor}(v, 1), \text{Neighbor}(v, 2), \dots$  until a query returns  $\perp$ . If this stage ends up making  $3n^{3/2}$  queries (say) without having processed every vertex, then it aborts and the algorithm moves on to the second stage. By Fact 2.2.1, if  $\kappa \leq \sqrt{n}$ , then this stage will not abort and the algorithm will have obtained  $G$  completely; it can then  $(\kappa + 1)$ -color  $G$  (as in Lemma 2.2.5) and terminate, skipping the second stage.

In the second stage, we know that  $\kappa > \sqrt{n}$ . The algorithm now uses a random coloring  $\psi$  to construct an  $(\ell, s, \lambda)$ -LDP of  $G$  using the “guess”  $k = \sqrt{n}$ , with  $s = \Theta(\varepsilon^{-2}n \log n)$  and  $\ell, \lambda$  given by Equation (2.6). To produce each subgraph  $G_i$  in the LDP, the algorithm simply makes all possible queries  $\text{Pair}(u, v)$  where  $\psi(u) = \psi(v)$ . W.h.p., the number of queries made is at most

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

where the first inequality uses Item (ii) of Theorem 2.2.6. We can enforce this bound in the worst case by aborting if it is violated.

Clearly,  $k \leq 2\kappa$ , so Item (i) of Theorem 2.2.6 applies and by the discussion after Definition 2.2.7, the algorithm uses  $(1 + O(\varepsilon))\kappa$  colors. Setting  $\varepsilon = 1/\log n$ , this number is at most  $\kappa + o(\kappa)$  and the overall number of queries remains  $\tilde{O}(n^{3/2})$ , as required.  $\square$

**Query Complexity Lower Bounds.** We complement our algorithmic results in the query model with lower bounds. Recall that the above algorithm produces a  $\kappa(1 + o(1))$ -coloring while making at most  $\tilde{O}(n^{3/2})$  queries, without needing to know  $\kappa$  in advance. Here, we shall prove that the number of colors cannot be improved to  $\kappa + 1$ : that would preclude sublinear complexity. In fact, we prove more general results, similar in spirit to the streaming lower bounds from Section 2.2.6. For the query lower bounds, we use another family of gadget graphs.

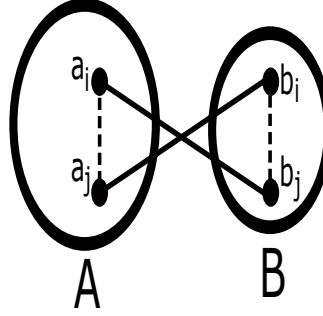


Figure 2.3: Gadget graph for proving query lower bounds

**Definition 2.2.18.** Given a large integer  $p$  (a size parameter), the gadgets for that size are  $(2p + 1)$ -vertex graphs on vertex set  $A \uplus B$ , where  $A = \{a_1, \dots, a_{p+1}\}$  and  $B = \{b_1, \dots, b_p\}$ . Let  $H$  be the graph consisting of a clique on  $A$  and a clique on  $B$ , with no edges between  $A$  and  $B$ . For  $1 \leq i < j \leq p$ , let  $H_{ij}$  be a graph on the same vertex set obtained by slightly modifying  $H$  as follows (see Figure 2.3):

$$E(H_{ij}) = E(H) \setminus \{ \{a_i, a_j\}, \{b_i, b_j\} \} \cup \{ \{a_i, b_j\}, \{a_j, b_i\} \}. \quad (2.9)$$

Notice that the vertex  $a_{p+1}$  is not touched by any of these modifications. The relevant properties of these gadget graphs are as follows.

**Lemma 2.2.19.** For all  $1 \leq i < j \leq p$ ,  $\kappa(H_{ij}) = p - 1$ , whereas the chromatic number  $\chi(H) = p + 1$ .

*Proof.* The claim about  $\chi(H)$  is immediate.

Consider a particular graph  $H_{ij}$ . The subgraph induced by  $A \setminus \{a_i\}$  is a  $p$ -clique, so  $\kappa(H_{ij}) \geq p - 1$ .

Now consider the following ordering  $\triangleleft$  for  $H_{ij}$ :  $B \triangleleft a_i \triangleleft A \setminus \{a_i\}$ , where the order within each set is arbitrary. For each  $v \in B$ ,  $\text{odeg}_{H_{ij}, \triangleleft}(v) \leq \deg(v) = p - 1$ . For each  $v \in A \setminus \{a_i\}$ ,  $\text{odeg}_{H_{ij}, \triangleleft}(v) \leq |A \setminus \{a_i\}| - 1 = p - 1$ . Finally,  $a_i$  has exactly  $p - 1$

neighbors in  $A \setminus \{a_i\}$  (by construction,  $a_j$  is not a neighbor), so  $\text{odeg}_{H_{ij}, \triangleleft}(a_i) = p - 1$ . By Lemma 2.2.4, it follows that  $\kappa(H_{ij}) \leq p - 1$ .  $\square$

Our proofs will use these gadget graphs in reductions from a pair of basic problems in decision tree complexity. Consider inputs that are vectors in  $\{0, 1\}^N$ : let  $\mathbf{0}$  denote the all-zero vector and, for  $i \in [N]$ , let  $\mathbf{e}_i$  denote the vector whose  $i$ th entry is 1 while all other entries are 0. Let  $\text{UNIQUE-OR}_N$  and  $\text{UNIQUE-FIND}_N$  denote the following partial functions on  $\{0, 1\}^N$ :

$$\text{UNIQUE-OR}_N(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} = \mathbf{0}, \\ 1, & \text{if } \mathbf{x} = \mathbf{e}_i, \text{ for } i \in [N], \\ \star, & \text{otherwise;} \end{cases}$$

$$\text{UNIQUE-FIND}_N(\mathbf{x}) = \begin{cases} i, & \text{if } \mathbf{x} = \mathbf{e}_i, \text{ for } i \in [N], \\ \star, & \text{otherwise.} \end{cases}$$

Informally, these problems capture, respectively, the tasks of (a) determining whether there is a needle in a haystack under the promise that there is at most one needle, and (b) finding a needle in a haystack under the promise that there is exactly one needle. Intuitively, solving either of these problems with high accuracy should require searching almost the entire haystack. Formally, let  $R_\delta^{\text{dt}}(f)$  denote the  $\delta$ -error randomized query complexity (a.k.a. decision tree complexity) of  $f$ . Elementary considerations of decision tree complexity lead to the bounds below (for a thorough discussion, including formal definitions, we refer the reader to the survey by Buhrman and de Wolf [51]).

**Fact 2.2.3.** *For all  $\delta \in (0, \frac{1}{2})$ , we have  $R_\delta^{\text{dt}}(\text{UNIQUE-OR}_N) \geq (1 - 2\delta)N$  and  $R_\delta^{\text{dt}}(\text{UNIQUE-FIND}_N) \geq (1 - \delta)N - 1$ .*  $\square$

With this setup, we turn to lower bounds of the first flavor.

**Lemma 2.2.20.** *Solving  $\text{GRAPH-DIST}(n, p, \lambda)$  in the general graph query model requires  $\Omega(n^2/\lambda^2)$  queries.*

*More precisely, there is a constant  $c > 0$  such that for every integer  $\lambda \geq 1$  and every sufficiently large integer  $p$ , there is a setting  $n = n(p, \lambda)$  for which every randomized query algorithm for  $\text{GRAPH-DIST}(n, p, \lambda)$  requires at least  $cn^2/\lambda^2$  queries in the worst case.*

*Proof.* We reduce from  $\text{UNIQUE-OR}_N$ , where  $N = \binom{p}{2}$ , using the following plan. Put  $n = (2p + 1)\lambda$ . Let  $\mathcal{C}$  be a query algorithm for  $\text{GRAPH-DIST}(n, p, \lambda)$ . Based on  $\mathcal{C}$ , we shall design a  $\frac{1}{3}$ -error algorithm  $\mathcal{A}$  for  $\text{UNIQUE-OR}_N$  that makes at most as many queries as  $\mathcal{C}$ . By Fact 2.2.3, this number of queries must be at least  $N/3 = \Omega(p^2) = \Omega(n^2/\lambda^2)$ .

As usual, we detail our reduction for  $\lambda = 1$ ; the Blow-up Lemma (Lemma 2.2.10) then handles general  $\lambda$ . By Lemma 2.2.19,  $H \in \mathcal{G}_1$  whereas each  $H_{ij} \in \mathcal{G}_2$  (cf. Definition 2.2.11, taking  $q = p$ ).

We now design  $\mathcal{A}$ . Let  $\mathbf{x} \in \{0, 1\}^N$  be the input to  $\mathcal{A}$ . Using a canonical bijection, let us index the bits of  $\mathbf{x}$  as  $x_{ij}$ , where  $1 \leq i < j \leq p$ . Algorithm  $\mathcal{A}$  simulates  $\mathcal{C}$  and outputs 1 iff  $\mathcal{C}$  decides that its input lies in  $\mathcal{G}_2$ . Since  $\mathcal{C}$  makes queries to a graph, we shall design an oracle for  $\mathcal{C}$  whose answers, based on query answers for input  $\mathbf{x}$  to  $\mathcal{A}$ , will implicitly define a graph on vertex set  $V := A \uplus B$ , as in Definition 2.2.18. The oracle answers queries as follows.

- For  $i, j \in [p]$ , it answers  $\text{Pair}(a_i, a_j)$  and  $\text{Pair}(b_i, b_j)$  with  $1 - x_{ij}$ .
- For  $i, j \in [p]$ , it answers  $\text{Pair}(a_i, b_j)$  and  $\text{Pair}(a_j, b_i)$  with  $x_{ij}$ .
- For  $i \in [p]$ , it answers  $\text{Pair}(a_{p+1}, a_i)$  with 1 and  $\text{Pair}(a_{p+1}, b_i)$  with 0.
- For  $i \in [p]$  and  $d \in [p - 1]$ , it answers  $\text{Neighbor}(a_i, d)$  with  $a_j$  if  $x_{ij} = 0$  and  $b_j$  if  $x_{ij} = 1$ , where  $j = d$  if  $d < i$ , and  $j = d + 1$  otherwise.
- For  $i, d \in [p]$ , it answers  $\text{Neighbor}(a_i, p)$  with  $a_{p+1}$  and  $\text{Neighbor}(a_{p+1}, d)$  with  $a_d$ .

- For  $i \in [p]$  and  $d \in [p - 1]$ , it answers  $\text{Neighbor}(b_i, d)$  with  $b_j$  if  $x_{ij} = 0$  and  $a_j$  if  $x_{ij} = 1$ , where  $j = d$  if  $d < i$ , and  $j = d + 1$  otherwise.
- For all other combinations of  $v \in V$  and  $d \in \mathbb{N}$ , it answers  $\text{Neighbor}(v, d) = \perp$ .

By inspection, we see that the graph defined by this oracle is  $H$  if  $\mathbf{x} = \mathbf{0}$  and is  $H_{ij}$  if  $\mathbf{x} = \mathbf{e}_{ij}$ . Furthermore, the oracle answers each query by making at most one query to the input  $\mathbf{x}$ . It follows that  $\mathcal{A}$  makes at most as many queries as  $\mathcal{C}$  and decides  $\text{UNIQUE-OR}_N$  with error at most  $\frac{1}{3}$ . This completes the proof for  $\lambda = 1$ .

To handle  $\lambda > 1$ , we modify the oracle in the natural way so that the implicitly defined graph is  $H^\lambda$  when  $\mathbf{x} = \mathbf{0}$  and  $H_{ij}^\lambda$  when  $\mathbf{x} = \mathbf{e}_{ij}$ . We omit the details, which are routine.  $\square$

As an immediate consequence of Lemma 2.2.20, we get the following query lower bounds.

**Theorem 2.2.21.** *Given query access to an  $n$ -vertex graph  $G$ , succeeding with probability at least  $2/3$  at either of the following tasks requires  $\Omega(n^2/\lambda^2)$  queries, where  $\lambda \geq 1$  is an integer parameter:*

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

We now prove a lower bound of the second flavor, where the algorithm knows  $\kappa$  in advance.

**Theorem 2.2.22.** *Given an integer  $\kappa$  and query access to an  $n$ -vertex graph  $G$  with  $\kappa(G) = \kappa$ , an algorithm that, with probability  $2/3$ , produces a proper  $(\kappa + \lambda)$ -coloring of  $G$  must make  $\Omega(n^2/\lambda^2)$  queries.*

*Proof.* We focus on the case  $\lambda = 1$ ; the general case is handled by the Blow-up Lemma, as usual.

Let  $\mathcal{C}$  be an algorithm for the coloring problem. We design an algorithm  $\mathcal{A}$  for  $\text{UNIQUE-FIND}_N$ , where  $N = \binom{p}{2}$ , using the same reduction as in Lemma 2.2.20, changing the post-processing logic as follows:  $\mathcal{A}$  outputs  $(i, j)$  as its answer to  $\text{UNIQUE-FIND}_N(\mathbf{x})$ , where  $1 \leq i < j \leq p$  is such that  $a_i$  and  $a_j$  are colored the same by  $\mathcal{C}$ .

To prove the correctness of this reduction, note that when  $\mathbf{x} = \mathbf{e}_{ij}$ , the graph defined by the simulated oracle is  $H_{ij}$  and  $\kappa(H_{ij}) = p - 1$  (Lemma 2.2.19). Suppose that  $\mathcal{C}$  is successful, which happens with probability at least  $2/3$ . Then  $\mathcal{C}$  properly  $p$ -colors  $H_{ij}$ . Recall that  $V(H_{ij}) = A \uplus B$ , where  $|A| = p + 1$ ; there must therefore be a color repetition within  $A$ . The only two non-adjacent vertices inside  $A$  are  $a_i$  and  $a_j$ , so  $\mathcal{A}$  correctly answers  $(i, j)$ . By Fact 2.2.3,  $\mathcal{A}$  must make  $\Omega(N) = \Omega(p^2)$  queries.  $\square$

**MPC and Congested Clique Models.** In the Massively Parallel Computations (MPC) model of Beame et al. [36], an input of size  $m$  is distributed adversarially among  $p$  processors, each of which has  $S$  bits of working memory: here,  $p$  and  $S$  are  $o(m)$  and, ideally,  $p \approx m/S$ . Computation proceeds in synchronous rounds: in each round, a processor carries out some local computation (of arbitrary time complexity) and then communicates with as many of the other processors as desired, provided that each processor sends and receives no more than  $S$  bits per round. The primary goal in solving a problem is to minimize the number of rounds.

When the input is an  $n$ -vertex graph, the most natural and widely studied setting of MPC is  $S = \tilde{O}(n)$ , which enables each processor to hold some information about every vertex; this makes many graph problems tractable. Since the input size  $m$  is potentially  $\Omega(n^2)$ , it is reasonable to allow  $p = n$  many processors. Note that the input is just a collection of edges, distributed adversarially among these processors, subject to the memory constraint.

**Theorem 2.2.23.** *There is a randomized  $O(1)$ -round MPC algorithm that, given an  $n$ -vertex graph  $G$ , outputs a  $\kappa(1 + o(1))$ -coloring of  $G$  with high probability. The algorithm uses  $n$  processors, each with  $O(n \log^2 n)$  bits of memory.*

*Proof.* Our algorithm will use  $n$  processors, each assigned to one vertex. If  $|E(G)| = O(n \log n)$ , then all of  $G$  can be collected at one processor in a single round using  $|E(G)| \cdot 2\lceil \log n \rceil = O(n \log^2 n)$  bits of communication and the problem is solved trivially. Therefore, we may as well assume that  $|E(G)| = \omega(n \log n)$ , which implies  $\kappa = \omega(\log n)$ , by Fact 2.2.1. We shall first give an algorithm assuming that  $\kappa$  is known *a priori*. Our final algorithm will be a refinement of this preliminary one.

*Preliminary algorithm.* Take  $k = \kappa$ . Each processor chooses a random color for its vertex, implicitly producing a partition  $(G_1, \dots, G_\ell)$  that is, w.h.p., an  $(\ell, s, \lambda)$ -LDP; we take  $\ell, \lambda$  as in eq. (2.6),  $s = \Theta(\varepsilon^{-2} n \log n)$ , and  $\varepsilon = (k^{-1} \log n)^{1/4}$ . Note that  $\varepsilon = o(1)$ . In Round 1, each processor sends its chosen color to all others—this is  $O(n \log n)$  bits of communication per machine—and as a result every processor learns which of its vertex’s incident edges are monochromatic. Now each color  $i \in [\ell]$  is assigned a unique machine  $M_i$  and, in Round 2, all edges in  $G_i$  are sent to  $M_i$ . Each  $M_i$  then locally computes a  $(\kappa(G_i) + 1)$ -coloring of  $G_i$  using a palette disjoint from those of other  $M_i$ s; by the discussion following Definition 2.2.7, this colors  $G$  using at most  $(1 + O(\varepsilon))\kappa = \kappa + o(\kappa)$  colors.

The communication in Round 2 is bounded by  $\max_i |E(G_i)| \cdot 2\lceil \log n \rceil$ . By Fact 2.2.1, items (i) and (ii) of Theorem 2.2.6, and eq. (2.6), the following holds w.h.p. for each  $i \in [\ell]$ :

$$|E(G_i)| \leq \kappa(G_i)|V(G_i)| \leq \frac{\kappa + \lambda}{\ell} \cdot \frac{2n}{\ell} \leq \frac{4n\kappa}{\ell^2} \leq \frac{4nk}{(2nk/s)^2} = \frac{O(\varepsilon^{-2} n \log n)^2}{nk} = O(n \log n). \quad (2.10)$$

Thus, the communication per processor in Round 2 is  $O(n \log^2 n)$  bits.

*Final algorithm.* When we don’t know  $\kappa$  in advance, we can make geometrically spaced guesses  $k$ , as in Section 2.2.5. In Round 1, we choose a random coloring for each such  $k$ . In Round 2, we determine the quantities  $|E(G_i)|$  for each  $k$  and each subgraph  $G_i$  and thereby determine the smallest  $k$  such that eq. (2.10) holds for every  $G_i$  corresponding to

this  $k$ . We then run Round 3 for only this one  $k$ , replicating the logic of Round 2 of the preliminary algorithm.

Correctness is immediate. We turn to bounding the communication cost. For Round 3, the previous analysis shows that the communication per processor is  $O(n \log^2 n)$  bits. For Rounds 1 and 2, let us consider the communication involved for each guess  $k$ : since each randomly-chosen color and each cardinality  $|E(G_i)|$  can be described in  $O(\log n)$  bits, each processor sends and receives at most  $O(n \log n)$  bits per guess. This is a total of  $O(n \log^2 n)$  bits, as claimed.  $\square$

The Congested-Clique model [133] is a well established model of distributed computing for graph problems. In this model, there are  $n$  nodes, each of which holds the local neighborhood information (i.e., the incident edges) of one vertex of the input graph  $G$ . In each round, every pair of nodes may communicate, whether or not they are adjacent in  $G$ , but the communication is restricted to  $O(\log n)$  bits. There is no constraint on a node's local memory. The goal is to minimize the number of rounds.

Behnezhad et al. [38] built on results of Lenzen [127] to show that any algorithm in the *semi-MPC model*—defined as MPC with space per machine being  $O(n \log n)$  bits—can be simulated in the Congested Clique model, preserving the round complexity up to a constant factor. Based on this, we obtain the following result.

**Theorem 2.2.24.** *There is a randomized  $O(1)$ -round algorithm in the Congested Clique model that, given a graph  $G$ , w.h.p. finds a  $(\kappa + O(\kappa^{3/4} \log^{1/2} n))$ -coloring. For  $\kappa = \omega(\log^2 n)$ , this gives a  $\kappa(1 + o(1))$ -coloring.*  $\square$

*Proof.* We cannot directly use our algorithm in Theorem 2.2.23 because it is not a semi-MPC algorithm: it uses  $O(n \log^2 n)$  bits of space per processor, rather than  $O(n \log n)$ . However, with a more efficient implementation of Round 1, a more careful analysis of Round 2, and a slight tweak of parameters for Round 3, we can improve the commu-

nication (hence, space) bounds to  $O(n \log n)$ , whereupon the theorem of Behnezhad et al. [38] completes the proof.

For Round 3, the tweak is to set  $\varepsilon = (k^{-1} \log^2 n)^{1/4}$  but otherwise replicate the logic of the final algorithm from Theorem 2.2.23. With this higher value of  $\varepsilon$ , the bound from eq. (2.10) improves to  $|E(G_i)| = O(n)$ . Therefore the per-processor communication in Round 3 is only  $O(n \log n)$  bits. The number of colors used is, w.h.p., at most  $(1 + O(\varepsilon))\kappa = \kappa + O(\kappa^{3/4} \log^{1/2} n)$ .

For a tighter analysis of the communication cost of Round 2, note that, for a particular guess  $k$ , there is a corresponding  $\ell$  given by eq. (2.6) such that each processor need only send/receive  $\ell$  cardinalities  $|E(G_i)|$ , each of which can be described in  $O(\log n)$  bits. Consulting eq. (2.6), we see that  $\ell = O(n^2/s) = O(n/\log n)$ . Therefore, summing over all  $O(\log n)$  choices of  $k$ , each processor communicates at most

$$O(n/\log n) \cdot O(\log n) \cdot O(\log n) = O(n \log n) \text{ bits.}$$

Round 1 appears problematic at first, since there are  $O(\log n)$  many random colorings to be chosen, one for each guess  $k$ . However, note that these colorings need not be independent. Therefore, we can choose just one random  $\lceil \log n \rceil$ -bit “master color”  $\phi(v)$  for each vertex  $v$  and derive the random colorings for the various guesses  $k$  by using only appropriate length prefixes of  $\phi(v)$ . This ensures that each processor only communicates  $O(n \log n)$  bits in Round 1.  $\square$

***Distributed Coloring in the LOCAL Model.*** In the LOCAL model, each node of the input graph  $G$  hosts a processor that knows only its own neighborhood. The processors operate in synchronous rounds, during which they can send and receive messages of arbitrary length to and from their neighbors. The processors are allowed unbounded local computation in each round. The key complexity measure is *time*, defined as the number of rounds used by

an algorithm (expected number, for a randomized algorithm) on a worst-case input.

Graph coloring in the LOCAL model is very heavily studied and is one of *the* central problems in distributed algorithms. Here, our focus is on algorithms that properly color the input graph  $G$  using a number of colors that depends on  $\alpha := \alpha(G)$ , the arboricity of  $G$ . Recall that  $\alpha \leq \kappa \leq 2\alpha - 1$  (Fact 2.2.2). Unlike in previous sections, our results will give big- $O$  bounds on the number of colors, so we may as well state them in terms of  $\alpha$  (following established tradition in this line of work) rather than  $\kappa$ . Our focus will be on algorithms that run in *sublogarithmic* time, while using not too many colors. See Section 4.2.1 for a quick summary of other interesting parameter regimes and Barenboim and Elkin [31] for a thorough treatment of graph coloring in the LOCAL model.

Kothapalli and Pemmaraju [124] gave an  $O(k)$ -round algorithm that uses  $O(\alpha n^{1/k})$  colors, for all  $k$  with  $2 \log \log n \leq k \leq \sqrt{\log n}$ . We give a new coloring algorithm that, in particular, extends the range of  $k$  to which such a time/quality tradeoff applies: for  $k \in [\omega(\sqrt{\log n}), o(\log n)]$ , we can compute an  $O(\alpha n^{1/k} \log n)$ -coloring in  $O(k)$  rounds.

Our algorithm uses our LDP framework to split the input graph into parts with logarithmic degeneracy (hence, arboricity) and then invokes an algorithm of Barenboim and Elkin. The following theorem records the key properties of their algorithm.

**Lemma 2.2.25** (Thm 5.6 of Barenboim and Elkin [29]). *There is a deterministic distributed algorithm in the LOCAL model that, given an  $n$ -vertex graph  $G$ , an upper bound  $b$  on  $\alpha(G)$ , and a parameter  $t$  with  $2 < t \leq O(\sqrt{n/b})$ , produces an  $O(tb^2)$ -coloring of  $G$  in time  $O(\log_t n + \log^* n)$ .  $\square$*

Here is the main result of this section.

**Theorem 2.2.26.** *There is a randomized distributed algorithm in the LOCAL model that, given an  $n$ -vertex graph  $G$ , an estimate of its arboricity  $\alpha$  up to a constant factor, and a parameter  $t$  such that  $2 < t \leq O(\sqrt{n/\log n})$ , produces an  $O(t\alpha \log n)$ -coloring of  $G$  in time  $O(\log_t n + \log^* n)$ .*

*Proof.* To simplify the presentation, we assume that  $\alpha = \alpha(G)$ . We assume that every node (vertex) knows  $n$  and  $\alpha$ . Consider a  $(\ell, s, \lambda)$ -LDP of  $G$ , where we put  $s = Cn \log n$ , for some large constant  $C$ , as in Theorem 2.2.6. This setting of  $s$  gives  $\ell = O(\alpha/\log n)$ . First, each vertex  $v$  chooses a color  $\psi(v)$  uniformly at random from  $[\ell]$ . Next, we need to effectively “construct” the blocks  $G_i$ , for each  $i \in [\ell]$ . This is straightforwardly done in a single round: each vertex  $v$  sends  $\psi(v)$  to all its neighbors.

At this point, each vertex  $v$  knows its neighbors in the block  $G_{\psi(v)}$ . So it’s now possible to run a distributed algorithm on each  $G_i$ . We invoke the algorithm in Lemma 2.2.25. The algorithm needs each vertex  $v$  to know an upper bound  $b_i$  on  $\alpha(G_i)$ , where  $i = \psi(v)$ . A useful upper bound of  $b_i = O(\log n)$ , which holds w.h.p., is given by item (i) of Theorem 2.2.6.

By Lemma 2.2.25, each  $G_i$  can be colored using  $O(t \log^2 n)$  colors, within another  $O(\log_t n + \log^* n)$  rounds, since  $2 < t \leq O(\sqrt{n/\log n})$ . Using disjoint palettes for the distinct blocks, the total number of colors used for  $G$  is at most  $\ell \cdot O(t \log^2 n) = O(t\alpha \log n)$ , as required.  $\square$

The particular form of the tradeoff stated in Table 2.1 is obtained by setting  $t = n^{1/k}$  (for some  $k \geq 3$ ) in the above theorem.

**Corollary 2.2.27.** *There is a randomized LOCAL algorithm that, given graph  $G$ , estimate  $\alpha \approx \alpha(G)$ , and a parameter  $k$  with  $2 < n^{1/k} \leq O(\sqrt{n/\log n})$ , finds an  $O(\alpha n^{1/k} \log n)$ -coloring of  $G$  in time  $O(k + \log^* n)$ .*  $\square$

### 2.2.8. A Combinatorial Lower Bound

Finally, we explore a connection between degeneracy based coloring and the *list coloring problem*. In the latter problem, each vertex has a list of colors and the goal is to find a corresponding list coloring—i.e., a proper coloring of the graph where each vertex receives a color from its list—or to report that none exists. Assadi et al. [17] proved a beautiful

*Palette Sparsification Theorem*, a purely graph-theoretic result that connects the  $(\Delta + 1)$ -coloring problem to the list coloring problem.

Define a graph  $G$  to be  $[\ell, r]_\delta$ -randomly list colorable (briefly,  $[\ell, r]_\delta$ -RLC) if choosing  $r$  random colors per vertex, independently and uniformly without replacement from the palette  $[\ell]$ , permits a list coloring with probability at least  $1 - \delta$  using these chosen lists.<sup>7</sup> Their theorem can be paraphrased as follows.

**Fact 2.2.4** (Assadi et al. [17], Theorem 1). *There exists a constant  $c$  such that every  $n$ -vertex graph  $G$  is  $[\Delta(G) + 1, c \log n]_{1/n}$ -RLC.*  $\square$

Indeed, this theorem is the basis of the various coloring results in their work. Let us outline how things work in the streaming model, focusing on the space usage. Given an input graph  $G$  that is promised to be  $[\ell, r]_{1/3}$ -RLC, for some parameters  $\ell, r$  that may depend on  $G$ , we sample  $r$  random colors from  $[\ell]$  for each vertex before reading the input. Chernoff bounds imply that the *conflict graph*—the subgraph of  $G$  consisting only of edges between vertices whose color lists intersect—is of size  $O(|E(G)|r^2/\ell)$ , w.h.p.. Using  $|E(G)| \leq n\Delta/2$ , taking  $\ell = \Delta + 1$  and  $r = O(\log n)$  bounds this size by  $\tilde{O}(n)$ , so a semi-streaming space bound suffices to collect the entire conflict graph. (For full details, see Lemma 4.1 in [17].) Finding a list coloring of the conflict graph (which exists with probability at least  $2/3$ ) yields an  $\ell$ -coloring of  $G$ .

For a similar technique to work in our setting, we would want  $\ell \approx \kappa$ . Recalling that  $|E(G)| \leq n\kappa$ , for the space usage to be  $\tilde{O}(n)$ , we need  $r = O(\text{polylog } n)$ . This raises the following combinatorial question: what is the smallest  $\lambda$  for which we can guarantee that every graph is  $[\kappa + \lambda, O(\text{polylog } n)]_{1/3}$ -RLC?

By the discussion above, our streaming lower bound in Theorem 2.2.15 already tells us that such a result is not possible with  $\lambda = O(\kappa^{\frac{1}{2}-\gamma})$ . Our final result (Theorem 2.2.29 below) proves that we can say much more.

<sup>7</sup>When  $r \geq \ell$ , this procedure simply produces the list  $[\ell]$  for every vertex.

Let  $J_{n,t}$  denote the graph  $K_t + \overline{K}_{n-t}$ , i.e., the graph join of a  $t$ -clique and an  $(n-t)$ -sized independent set. More explicitly,

$$J_{n,t} = (A \uplus B, E), \quad \text{where } |A| = t, |B| = n-t, E = \{\{u, v\} : u \in A, v \in A \cup B, u \neq v\}. \quad (2.11)$$

**Lemma 2.2.28.** *For integers  $0 < r \leq t < n$ , if  $J_{n,t}$  is  $[\kappa + \kappa/r, r]_\delta$ -RLC, then  $\delta \geq 1 - r^n/(r+1)^{n-t}$ .*

*Proof.* Take a graph  $J_{n,t}$  with vertices partitioned into  $A$  and  $B$  as in eq. (2.11). An ordering with  $B \triangleleft A$  shows that  $\kappa = \kappa(J_{n,t}) = t$ . We claim that for every choice of colors lists for vertices in  $A$ , taken from the palette  $[t + t/r]$ , the probability that the chosen lists for  $B$  permit a proper list coloring is at most  $p := r^n/(r+1)^{n-t}$ . This will prove that  $\delta \geq 1 - p$ .

To prove the claim, consider a particular choice of lists for  $A$ . Fix a partial coloring  $\psi$  of  $A$  consistent with these lists. If  $\psi$  is not proper, there is nothing to prove. Otherwise, since  $A$  induces a clique,  $\psi$  must assign  $t$  distinct colors to  $A$ . In order for a choice of lists for  $B$  to permit a proper extension of  $\psi$  to the entire graph, every vertex of  $B$  must sample a color from the remaining  $t/r$  colors in the palette. Since  $r$  colors are chosen per vertex, this event has probability at most

$$\left(r \cdot \frac{t/r}{t + t/r}\right)^{|B|} = \left(\frac{r}{r+1}\right)^{n-t}.$$

The claimed upper bound on  $p$  now follows by a union bound over the  $r^t$  possible partial colorings  $\psi$ . □

This easily leads to our combinatorial lower bound, given below. In reading the theorem statement, note that the restriction on edge density *strengthens* the theorem.

**Theorem 2.2.29.** *Let  $n$  be sufficiently large and let  $m$  be such that  $n \leq m \leq n^2/\log^2 n$ . If every  $n$ -vertex graph  $G$  with  $\Theta(m)$  edges is  $[\kappa(G) + \lambda, c \log n]_{1/3}$ -RLC for some parameter*

$\lambda$  and some constant  $c$ , then we must have  $\lambda > \kappa(G)/(c \log n)$ .

*Proof.* Suppose not. Put  $t = \lceil m/n \rceil$ ,  $r = c \log n$ , and consider the graph  $J_{n,t}$  defined in eq. (2.11). By the bounds on  $m$ ,  $|E(J_{n,t})| = t(t-1)/2 + t(n-t) = \Theta(nt) = \Theta(m)$ . Put  $\kappa := \kappa(J_{n,t})$ . By assumption,  $J_{n,t}$  is  $[\kappa + \kappa/r, r]$ -RLC, so Lemma 2.2.28 implies that

$$\frac{2}{3} \leq \frac{r^n}{(r+1)^{n-t}} = \left(1 - \frac{1}{r+1}\right)^n (r+1)^t \leq \exp\left(-\frac{n}{r+1} + t \ln(r+1)\right).$$

Since  $t = O(n/\log^2 n)$  and  $r = c \log n$ , this is a contradiction for sufficiently large  $n$ .  $\square$

We remark that the above result rules out the possibility of using a palette sparsification theorem along the lines of Assadi et al. [17] to obtain a semi-streaming coloring algorithm that uses fewer colors than Algorithm 1 (with the setting  $\varepsilon = 1/\log n$ ).

More generally, suppose we were willing to tolerate a weaker notion of palette sparsification by sampling  $O(\log^d n)$  colors per vertex, for some  $d \geq 1$ : this would increase the space complexity of an algorithm based on such sparsification by a  $\text{polylog } n$  factor. By Lemma 2.2.28, arguing as in Theorem 2.2.29, we would need to spend at least  $\kappa + \kappa/\Theta(\log^d n)$  colors. This is no better than the number of colors obtained using Algorithm 1 with the setting  $\varepsilon = 1/\log^d n$ , which still maintains semi-streaming space. In fact, palette sparsification does not immediately guarantee a post-processing runtime that is better than exponential, because we need to color the conflict graph in post-processing. Meanwhile, recall that Algorithm 1 has  $\tilde{O}(n)$  post-processing time via a straightforward greedy algorithm. Furthermore, since there exist “hard” graphs  $J_{n,t}$  at all edge densities from  $\Theta(n)$  to  $\Theta(n^2/\log^2 n)$ , we cannot even hope for a semi-streaming palette-sparsification-based algorithm that might work only for sparse graphs or only for dense graphs.

### 2.2.9. Subsequent works

Subsequently, a number of works studied streaming graph coloring from various angles. Alon and Assadi [8] explored palette sparsification under several settings of palette size

and number of sampled colors. Their results implied that there is a palette-sparsification-based semi-streaming algorithm for  $\kappa(1 + o(1))$ -coloring, although it does not guarantee polynomial time post-processing. Their palette sparsification theorems also implied semi-streaming algorithms for coloring triangle-free graphs and  $(\deg + 1)$ -coloring, where every vertex  $v$  gets a color list  $[\deg(v) + 1]$ . Very recently, Halldórsson [96] gave a palette-sparsification-based semi-streaming algorithm for  $(\deg + 1)$ -coloring that works even when the color list of node  $v$  can be any arbitrary list of  $(\deg(v) + 1)$  colors, not just  $[\deg(v) + 1]$ . Bhattacharya et al. [45] showed that verifying whether a given vertex-coloring (streamed with the input edges) is proper, does not admit any sublinear-space algorithm. They also gave some interesting algorithms for this problem for random-order streams. Assadi, Chen, and Sun [16] proved that  $(\Delta + 1)$ -coloring has no non-trivial one-pass *deterministic* semi-streaming algorithm: any such algorithm requires  $\exp(\Delta^{\Omega(1)})$  colors. Again, any one-pass deterministic algorithm using  $O(n^{1+\alpha})$  space requires  $\Omega(\Delta^{1/(2\alpha)})$  colors. Further, they gave a deterministic 2-pass  $O(\Delta^2)$ -coloring semi-streaming algorithm and extended it to an  $O(\log n)$ -pass  $O(\Delta)$ -coloring. Furthermore, Assadi, Kumar, and Mittal [19] surprisingly proved Brooks' theorem in the semi-streaming model: they showed that any connected graph which is not an odd cycle or a clique admits a  $\Delta$ -coloring algorithm in this setting. Finally, in a joint work with A. Chakrabarti and M. Stoeckl [61], we studied graph coloring in the adversarially robust streaming setting, of which give an elaborate account in the next chapter.

---

## Chapter 3

---

# Adversarially Robust Streaming

Recall that a data streaming algorithm processes a long input sequence  $\sigma$ , while using space *sublinear* in the size of  $\sigma$ , to return an output from a set of valid outputs based on  $\sigma$ . For most—though not all—problems of interest, a streaming algorithm *needs* to be randomized in order to achieve sublinear space. Observe that most algorithms given in Chapter 2 are randomized. For a randomized algorithm, the standard correctness requirement is that for each possible fixed input stream it return a valid answer with high probability. A burgeoning body of work—much of it very recent [21, 39–41, 50, 100, 115, 169] but traceable back to [97]—addresses streaming algorithms that seek an even stronger correctness guarantee: they need to produce valid answers with high probability even when working with an input generated by an active adversary. There is compelling motivation from practical applications for seeking this stronger guarantee: for instance, consider a user continuously interacting with a database and choosing future queries based on past answers received; or think of an online streaming or marketing service looking at a customer’s transaction history and recommending them products based on it.

We may view the operation of streaming algorithm  $\mathcal{A}$  as a game between a *solver*, who executes  $\mathcal{A}$ , and an *adversary*, who generates a “hard” input stream  $\sigma$ . The standard notion of  $\mathcal{A}$  having error probability  $\delta$  is that for every fixed  $\sigma$  that the adversary may choose, the

probability over  $\mathcal{A}$ 's random choices that it errs on  $\sigma$  is at most  $\delta$ . Since the adversary has to make their choice before the solver does any work, they are *oblivious* to the actual actions of the solver. In contrast to this, an *adaptive adversary* is not required to fix all of  $\sigma$  in advance, but can generate the elements (tokens) of  $\sigma$  incrementally, based on outputs generated by the solver as it executes  $\mathcal{A}$ . Clearly, such an adversary is much more powerful and can attempt to learn something about the solver's internal state in order to generate input tokens that are bad for the particular random choices made by  $\mathcal{A}$ . Indeed, such adversarial attacks are known to break many well known algorithms in the streaming literature [40,97]. Motivated by this, one defines a  $\delta$ -error *adversarially robust streaming algorithm* to be one where the probability that an adaptive adversary can cause the solver to produce an incorrect output at some point of time is at most  $\delta$ . Notice that a *deterministic* streaming algorithm (which, by definition, must always produce correct answers) is automatically adversarially robust.

Past work on such adversarially robust streaming algorithms has focused on statistical estimation problems and on sampling problems but, with the exception of [50], there has not been much study of graph theoretic problems. In this chapter, we mainly focus on graph coloring, a fundamental algorithmic problem on graphs. Recall that the goal is to efficiently process an input graph given as a stream of edges and assign colors to its vertices from a small palette so that no two adjacent vertices receive the same color. The main messages of our results are that (i) while there exist surprisingly efficient sublinear-space algorithms for coloring under standard streaming, it is provably harder to obtain adversarially robust solutions; but nevertheless, (ii) there *do* exist nontrivial sublinear-space robust algorithms for coloring.

To be slightly more detailed, suppose we must color an  $n$ -vertex input graph  $G$  that has maximum degree  $\Delta$ . Producing a coloring using only  $\chi(G)$  colors, where  $\chi(G)$  is the chromatic number, is NP-hard while producing a  $(\Delta + 1)$ -coloring admits a straightforward

greedy algorithm, given offline access to  $G$ . Producing a good coloring given only streaming access to  $G$  and sublinear (i.e.,  $o(n\Delta)$  bits of) space is a nontrivial problem and the subject of much recent research [8, 17, 43–45], including the breakthrough result of Assadi, Chen, and Khanna [17] that gives a  $(\Delta + 1)$ -coloring algorithm using only semi-streaming (i.e.,  $\tilde{O}(n)$  bits of) space.<sup>1</sup> However, all of these algorithms were designed with only the standard, oblivious adversary setting in mind; an adaptive adversary can make all of them fail. This is the starting point for our exploration in this work.

## Section 3.1

### Motivation and Context

Graph streaming has become widely popular [137], especially since the advent of large and evolving networks including social media, web graphs, and transaction networks. These large graphs are regularly mined for knowledge and such knowledge often informs their future evolution. Therefore, it is important to have adversarially robust algorithms for working with these graphs. Yet, the recent explosion of interest in robust algorithms has not focused much on graph problems. We now quickly recap some history.

Two influential works [97, 140] identified the challenge posed by adaptive adversaries to sketching and streaming algorithms. In particular, Hardt and Woodruff [97] showed that many statistical problems, including the ubiquitous one of  $\ell_2$ -norm estimation, do not admit adversarially robust linear sketches of sublinear size. Recent works have given a number of positive results. Ben-Eliezer, Jayaram, Woodruff, and Yogev [40] considered such fundamental problems as distinct elements, frequency moments, and heavy hitters (these date back to the beginnings of the literature on streaming algorithms); for  $(1 \pm \varepsilon)$ -approximating a function value, they gave two generic frameworks that can “robustify” a standard streaming algorithm, blowing up the space cost by roughly the *flip number*  $\lambda_{\varepsilon, m}$ ,

<sup>1</sup>The notation  $\tilde{O}(\cdot)$  hides factors polylogarithmic in  $n$ .

defined as the maximum number of times the function value can change by a factor of  $1 \pm \varepsilon$  over the course of an  $m$ -length stream. For *insertion-only streams* and monotone functions,  $\lambda_{\varepsilon, m}$  is roughly  $O(\varepsilon^{-1} \log m)$ , so this overhead is very small. Subsequent works [21, 100, 169] have improved this overhead with the current best-known one being  $O(\sqrt{\varepsilon \lambda_{\varepsilon, m}})$  [21].

For insertion-only graph streams, a number of well-studied problems such as triangle counting, maximum matching size, and maximum subgraph density can be handled by the above framework because the underlying functions are monotone. For some problems such as counting connected components, there are simple deterministic algorithms that achieve an asymptotically optimal space bound, so there is nothing new to say in the robust setting. For graph sparsification, [50] showed that the Ahn–Guha sketch [4] can be made adversarially robust with a slight loss in the quality of the sparsifier. Thanks to efficient adversarially robust sampling [41, 50], many sampling-based graph algorithms should yield corresponding robust solutions without much overhead. For problems calling for Boolean answers, such as testing connectivity or bipartiteness, achieving low error against an oblivious adversary automatically does so against an adaptive adversary as well, since a sequence of correct outputs from the algorithm gives away no information to the adversary. This is a particular case of a more general phenomenon captured by the notion of pseudo-determinism, discussed at the end of this section.

Might it be that for all interesting data streaming problems, efficient standard streaming algorithms imply efficient robust ones? The above framework does not automatically give good results for *turnstile* streams, where each token specifies either an insertion or a deletion of an item, or for estimating non-monotone functions. In either of these situations, the flip number can be very large. As noted above, linear sketching, which is the preeminent technique behind turnstile streaming algorithms (including ones for graph problems), is vulnerable to adversarial attacks [97]. This does not quite provide a separation between standard and robust space complexities, since it does not preclude efficient non-linear solu-

tions. The very recent work [115] gives such a separation: it exhibits a function estimation problem for which the ratio between the adversarial and standard streaming complexities is as large as  $\tilde{\Omega}(\sqrt{\lambda_{\varepsilon, m}})$ , which is exponential upon setting parameters appropriately. However, their function is highly artificial, raising the important question: *Can a significant gap be shown for a natural streaming problem?*<sup>2</sup>

It is easy to demonstrate such a gap in graph streaming. Consider the problem of finding a spanning forest in a graph undergoing edge insertions and deletions. The celebrated Ahn–Guha–McGregor sketch [5] solves this in  $\tilde{O}(n)$  space, but this sketch is not adversarially robust. Moreover, suppose that  $\mathcal{A}$  is an adversarially robust algorithm for this problem. Then we can argue that the memory state of  $\mathcal{A}$  upon processing an unknown graph  $G$  must contain enough information to recover  $G$  entirely: an adversary can repeatedly ask  $\mathcal{A}$  for a spanning forest, delete all returned edges, and recurse until the evolving graph becomes empty. Thus, for basic information theoretic reasons,  $\mathcal{A}$  must use  $\Omega(n^2)$  bits of space, resulting in a quadratic gap between robust and standard streaming space complexities. Arguably, this separation is not very satisfactory, since the hardness arises from the turnstile nature of the stream, allowing the adversary to delete edges. Meanwhile, the [115] separation does hold for insert-only streams, but as we (and they) note, their problem is rather artificial.

**Hardness for Natural Problems.** We now make a simple, yet crucial, observation. Let MISSING-ITEM-FINDING (MIF) denote the problem where, given an evolving set  $S \subseteq [n]$ , we must be prepared to return an element in  $[n] \setminus S$  or report that none exists. When the elements of  $S$  are given as an input stream, MIF admits the following  $O(\log^2 n)$ -space solution against an oblivious adversary: maintain an  $\ell_0$ -sampling sketch [109] for the characteristic vector of  $[n] \setminus S$  and use it to randomly sample a valid answer. In fact, this solution extends to turnstile streams. Now suppose that we have an adversarially robust algorithm  $\mathcal{A}$

---

<sup>2</sup>This open question was explicitly raised in the STOC 2021 workshop *Robust Streaming, Sketching, and Sampling* [159].

for MIF, handling insert-only streams. Then, given the memory state of  $\mathcal{A}$  after processing an unknown set  $T$  with  $|T| = n/2$ , an adaptive adversary can repeatedly query  $\mathcal{A}$  for a missing item  $x$ , record  $x$ , insert  $x$  as the next stream token, and continue until  $\mathcal{A}$  fails to find an item. At that point, the adversary will have recorded (w.h.p.) the set  $[n] \setminus T$ , so he can reconstruct  $T$ . As before, by basic information theory, this reconstructability implies that  $\mathcal{A}$  uses  $\Omega(n)$  space.

This exponential gap between standard and robust streaming, based on well-known results, seems to have been overlooked—perhaps because MIF does not conform to the type of problems, namely estimation of real-valued functions, that much of the robust streaming literature has focused on. That said, though MIF is a natural problem and the hardness holds for insert-only streams, there is one important box that MIF does not tick: it is not important enough on its own and so does not command a serious literature. This leads us to refine the open question of [115] thus: *Can a significant gap be shown for a natural and well-studied problem with the hardness holding even for insertion-only streams?*

With this in mind, we return to graph problems, searching for such a gap. In view of the generic framework of [40] and follow-up works, we should look beyond estimating some monotone function of the graph with scalar output. What about problems where the output is a big *vector*, such as approximate maximum matching (not just its size) or approximate densest subgraph (not just the density)? It turns out that the *sketch switching* technique of [40] can still be applied: since we need to change the output only when the estimates of the associated numerical values (matching size and density, respectively) change enough, we can proceed as in that work, switching to a new sketch with fresh randomness that remains unrevealed to the adversary. This gives us a robust algorithm incurring only logarithmic overhead.

But graph coloring is different. As our Theorem 3.2.1 shows, it does exhibit a quadratic gap for the right setting of parameters and it is, without doubt, a heavily-studied problem,

even in the data streaming setting.

The above hardness of MIF provides a key insight into why graph coloring is hard; see Section 3.2.3.

**Connections with Other Work on Streaming Graph Coloring.** Graph coloring is, of course, a heavily-studied problem in theoretical computer science. For this discussion, we stick to streaming algorithms for this problem, which already has a significant literature [1, 8, 17, 43–45].

Although it is not possible to  $\chi(G)$ -color an input graph in sublinear space [1], as [17] shows, there is a semi-streaming algorithm that produces a  $(\Delta + 1)$ -coloring. This follows from their elegant *palette sparsification* theorem, which states that if each vertex samples roughly  $O(\log n)$  colors from a palette of size  $\Delta + 1$ , then there exists a proper coloring of the graph where each vertex uses a color only from its sampled list. Hence, we only need to store edges between vertices whose lists intersect. If the edges of  $G$  are independent of the algorithm’s randomness, then the expected number of such “conflict” edges is  $O(n \log^2 n)$ , leading to a semi-streaming algorithm. But note that an adaptive adversary can attack this algorithm by using a reported coloring to learn which future edges would definitely be conflict edges and inserting such edges to blow up the algorithm’s storage.

There are some other semi-streaming algorithms (in the standard setting) that aim for  $\Delta(1 + \varepsilon)$ -colorings. One is palette-sparsification based [8] and so, suffers from the above vulnerability against an adaptive adversary. Others [43, 44] are based on randomly partitioning the vertices into clusters and storing only intra-cluster edges, using pairwise disjoint palettes for the clusters.

Here, the semi-streaming space bound hinges on the random partition being likely to assign each edge’s endpoints to different clusters. This can be broken by an adaptive adversary, who can use a reported coloring to learn many vertex pairs that are intra-cluster and then insert new edges at such pairs.

Finally, we highlight an important theoretical question about sublinear algorithms for graph coloring: *Can they be made deterministic?* This was explicitly raised by Assadi [15] and, prior to this work, it was open whether, for  $(\Delta + 1)$ -coloring, *any* sublinear space bound could be obtained deterministically. Our Theorem 3.2.1 settles the deterministic space complexity of this problem, showing that even the weaker requirement of  $O(\Delta)$ -coloring forces  $\Omega(n\Delta)$  space, which is linear in the input size.

Parameterizing Theorem 3.2.1 differently, we see that a robust (in particular, a deterministic) algorithm that is limited to semi-streaming space must spend  $\tilde{\Omega}(\Delta^2)$  colors. A major remaining open question is whether this can be matched, perhaps by a deterministic semi-streaming  $O(\Delta^2)$ -coloring algorithm. In fact, it is not known how to get even a  $\text{poly}(\Delta)$ -coloring deterministically. Our algorithmic results, summarized in Theorem 3.2.2, make partial progress on this question. Though we do not obtain deterministic algorithms, we obtain adversarially robust ones, and we do obtain  $\text{poly}(\Delta)$ -colorings, though not all the way down to  $O(\Delta^2)$  in semi-streaming space.

**Other Related Work.** Pseudo-deterministic streaming algorithms [93] fall between adversarially robust and deterministic ones. Such an algorithm is allowed randomness, but for each particular input stream it must produce one fixed output (or output sequence) with high probability. Adversarial robustness is automatic, because when such an algorithm succeeds, it does not reveal any of its random bits through the outputs it gives. Thus, there is nothing for an adversary to base adaptive decisions on.

The well-trodden subject of dynamic graph algorithms deals with a model closely related to the adaptive adversary model: one receives a stream of edge insertions/deletions and seeks to maintain a solution after each update. There have been a few works on the  $\Delta$ -based graph coloring problem in this setting [46, 47, 102]. However, the focus of the dynamic setting is on optimizing the update *time* without any restriction on the space usage; this is somewhat orthogonal to the streaming setting where the primary goal is space

efficiency, and update time, while practically important, is not factored into the complexity.

## Section 3.2

# Adversarially Robust Coloring

### 3.2.1. Our Results and Contributions

We ask whether the graph coloring problem is inherently harder under an adversarial robustness requirement than it is for standard streaming. We answer this question affirmatively with the first major theorem in this work, which is the following (we restate the theorem with more detail and formality as Theorem 3.2.6).

**Theorem 3.2.1.** *A constant-error adversarially robust algorithm that processes a stream of edge insertions into an  $n$ -vertex graph and, as long as the maximum degree of the graph remains at most  $\Delta$ , maintains a valid  $K$ -coloring (with  $\Delta + 1 \leq K \leq n/2$ ) must use at least  $\Omega(n\Delta^2/K)$  bits of space.*

We spell out some immediate corollaries of this result because of their importance as conceptual messages.

- **Robust coloring using  $O(\Delta)$  colors.** In the setting of Theorem 3.2.1, if the algorithm is to use only  $O(\Delta)$  colors, then it must use  $\Omega(n\Delta)$  space. In other words, a sublinear-space solution is ruled out.
- **Robust coloring using semi-streaming space.** In the setting of Theorem 3.2.1, if the algorithm is to run in only  $\tilde{O}(n)$  space, then it must use  $\tilde{\Omega}(\Delta^2)$  colors.
- **Separating robust from standard streaming with a natural problem.** Contrast the above two lower bounds with the guarantees of the [17] algorithm, which handles the non-robust case. This shows that “maintaining an  $O(\Delta)$ -coloring of a graph” is a natural (and well-studied) algorithmic problem where, even for insertion-only

streams, the space complexities of the robust and standard streaming versions of the problem are well separated: in fact, the separation is roughly quadratic, by taking  $\Delta = \Theta(n)$ . This answers an open question of [115], as we explain in greater detail in Section 4.2.1.

- **Deterministic versus randomized coloring.** Since every deterministic streaming algorithm is automatically adversarially robust, the lower bound in Theorem 3.2.1 applies to such algorithms. In particular, this settles the deterministic complexity of  $O(\Delta)$ -coloring. Also, turning to semi-streaming algorithms, whereas a combinatorially optimal<sup>3</sup>  $(\Delta + 1)$ -coloring is possible using randomization [17], a deterministic solution must spend at least  $\tilde{\Omega}(\Delta^2)$  colors. These results address a broadly-stated open question of Assadi [15]; see Section 4.2.1 for details.

We prove the lower bound in Theorem 3.2.1 using a reduction from a novel two-player communication game that we call SUBSET-AVOIDANCE. In this game, Alice is given an  $a$ -sized subset of the universe  $[t]$ ;<sup>4</sup> she must communicate a possibly random message to Bob that causes him to output a  $b$ -sized subset of  $[t]$  that, with high probability, avoids Alice’s set completely. We give a fairly tight analysis of the communication complexity of this game, showing an  $\Omega(ab/t)$  lower bound, which is matched by an  $\tilde{O}(ab/t)$  deterministic upper bound. The SUBSET-AVOIDANCE problem is a natural one. We consider the definition of this game and its analysis—which is not complicated—to be additional conceptual contributions of this work; these might be of independent interest for future applications.

We complement our lower bound with some good news: we give a suite of upper bound results by designing adversarially robust coloring algorithms that handle several interesting parameter regimes. Our focus is on maintaining a valid coloring of the graph using  $\text{poly}(\Delta)$  colors, where  $\Delta$  is the current maximum degree, as an adversary inserts edges. In fact,

<sup>3</sup>If one must use at most  $f(\Delta)$  colors for some function  $f$ , the best possible function that always works is  $f(\Delta) = \Delta + 1$ .

<sup>4</sup>The notation  $[t]$  denotes the set  $\{1, 2, \dots, t\}$ .

some of these results hold even in a *turnstile model*, where the adversary might both add and delete edges. In this context, it is worth noting that the [17] algorithm also works in a turnstile setting.

**Theorem 3.2.2.** *There exist adversarially robust algorithms for coloring an  $n$ -vertex graph achieving the following tradeoffs (shown in Table 3.1) between the space used for processing the stream and the number of colors spent, where  $\Delta$  denotes the evolving maximum degree of the graph and, in the turnstile setting,  $m$  denotes a known upper bound on the stream length.*

| Model                  | Colors        | Space                         | Notes                                     |
|------------------------|---------------|-------------------------------|---|
| Insertion-only         | $O(\Delta^3)$ | $\tilde{O}(n)$                | $\tilde{O}(n\Delta)$ external random bits |
| Insertion-only         | $O(\Delta^k)$ | $\tilde{O}(n\Delta^{1/k})$    | any $k \in \mathbb{N}$                    |
| Strict Graph Turnstile | $O(\Delta^k)$ | $\tilde{O}(n^{1-1/k}m^{1/k})$ | constant $k \in \mathbb{N}$               |

Table 3.1: A summary of our adversarially robust coloring algorithms. A “strict graph turnstile” model requires the input to describe a simple graph at all times; see Section 3.2.2.

*In each of these algorithms, for each stream update or query made by the adversary, the probability that the algorithm fails either by returning an invalid coloring or aborting is at most  $1/\text{poly}(n)$ .*

We give a more detailed discussion of these results, including an explanation of the technical caveat noted in Table 3.1 for the  $O(\Delta^3)$ -coloring algorithm, in Section 3.2.3.

### 3.2.2. Preliminaries

**Defining Adversarial Robustness.** For the purposes of this paper, a “streaming algorithm” is always one-pass and we always think of it as working against an adversary. In the *standard streaming* setting, this adversary is oblivious to the algorithm’s actual run. This can be thought of as a special case of the setup we now introduce in order to define *adversarially robust streaming* algorithms.

Let  $\mathcal{U}$  be a universe whose elements are called tokens. A data stream is a sequence in  $\mathcal{U}^*$ . A data streaming problem is specified by a relation  $f \subseteq \mathcal{U}^* \times \mathcal{Z}$  where  $\mathcal{Z}$  is some output domain: for each input stream  $\sigma \in \mathcal{U}^*$ , a valid solution is any  $z \in \mathcal{Z}$  such that  $(\sigma, z) \in f$ . A randomized streaming algorithm  $\mathcal{A}$  for  $f$  running in  $s$  bits of space and using  $r$  random bits is formalized as a triple consisting of (i) a function  $\text{INIT}: \{0, 1\}^r \rightarrow \{0, 1\}^s$ , (ii) a function  $\text{PROCESS}: \{0, 1\}^s \times \mathcal{U} \times \{0, 1\}^r \rightarrow \{0, 1\}^s$ , and (iii) a function  $\text{QUERY}: \{0, 1\}^s \times \{0, 1\}^r \rightarrow \mathcal{Z}$ . Given an input stream  $\sigma = (x_1, \dots, x_m)$  and a random string  $R \in_R \{0, 1\}^r$ , the algorithm starts in state  $w_0 = \text{INIT}(R)$ , goes through a sequence of states  $w_1, \dots, w_m$ , where  $w_i = \text{PROCESS}(w_{i-1}, x_i, R)$ , and provides an output  $z = \text{QUERY}(w_m, R)$ . The algorithm is  $\delta$ -error in the standard sense if  $\Pr_R[(\sigma, z) \in f] \geq 1 - \delta$ .

To define adversarially robust streaming, we set up a game between two players: Solver, who runs an algorithm as above, and Adversary, who adaptively generates a stream  $\sigma = (x_1, \dots, x_m)$  using a next-token function  $\text{NEXT}: \mathcal{Z}^* \rightarrow \mathcal{U}$  as follows. With  $w_0, \dots, w_m$  as above, put  $z_i = \text{QUERY}(w_i, R)$  and  $x_i = \text{NEXT}(z_0, \dots, z_{i-1})$ . In words, Adversary is able to query the algorithm at each point of time and can compute an arbitrary *deterministic* function of the history of outputs provided by the algorithm to generate his next token. Fix (an upper bound on) the stream length  $m$ . Algorithm  $\mathcal{A}$  is  $\delta$ -error adversarially robust if

$$\forall \text{ function NEXT} : \Pr_R[\forall i \in [m] : ((x_1, \dots, x_i), z_i) \in f] \geq 1 - \delta.$$

In this work, we prove lower bounds for algorithms that are only required to be  $O(1)$ -error adversarially robust. On the other hand, the algorithms we design will achieve vanishingly small error of the form  $1/\text{poly}(m)$  and moreover, they will be able to detect when they are about to err and can abort at that point.

**Graph Streams and the Coloring Problem.** Throughout this paper, an *insert-only graph stream* describes an undirected graph on the vertex set  $[n]$ , for some fixed  $n$  that is known in advance, by listing its edges in some order: each token is an edge. A *strict graph turnstile*

*stream* describes an evolving graph  $G$  by using two types of tokens—INS-EDGE( $\{u, v\}$ ), which causes  $\{u, v\}$  to be added to  $G$ , and DEL-EDGE( $\{u, v\}$ ), which causes  $\{u, v\}$  to be removed—and satisfies the promises that each insertion is of an edge that was not already in  $G$  and that each deletion is of an edge that was in  $G$ . When we use the term “graph stream” without qualification, it should be understood to mean an insert-only graph stream, unless the context suggests that either flavor is acceptable.

In this context, a semi-streaming algorithm is one that runs in  $\tilde{O}(n) := O(n \text{ polylog } n)$  bits of space.

In the  $K$ -coloring problem, the input is a graph stream and a valid answer to a query is a vector in  $[K]^n$  specifying a color for each vertex such that no two adjacent vertices receive the same color. The quantity  $K$  may be given as a function of some graph parameter, such as the maximum degree  $\Delta$ . In reading the results in this paper, it will be helpful to think of  $\Delta$  as a growing but sublinear function of  $n$ , such as  $n^\alpha$  for  $0 < \alpha < 1$ . Since an output of the  $K$ -coloring problem is a  $\Theta(n \log K)$ -sized object, we think of a semi-streaming coloring algorithm running in  $\tilde{O}(n)$  space as having “essentially optimal” space usage.

**One-Way Communication Complexity.** In this work, we shall only consider a special kind of two-player communication game: one where all input belongs to the speaking player Alice and her goal is to induce Bob to produce a suitable output. Such a game,  $g$ , is given by a relation  $g \in \mathcal{X} \times \mathcal{Z}$ , where  $\mathcal{X}$  is the input domain and  $\mathcal{Z}$  is the output domain. In a protocol  $\Pi$  for  $g$ , Alice and Bob share a random string  $R$ . Alice is given  $x \in \mathcal{X}$  and sends Bob a message  $\text{msg}(x, R)$ . Bob uses this to compute an output  $z = \text{out}(\text{msg}(x, R))$ . We say that  $\Pi$  solves  $g$  to error  $\delta$  if  $\forall x \in \mathcal{X} : \Pr_R[(x, z) \in g] \geq 1 - \delta$ . The communication cost of  $\Pi$  is  $\text{cost}(\Pi) := \max_{x, R} \text{length}(\text{msg}(x, R))$ . The (one-way, randomized, public-coin)  $\delta$ -error communication complexity of  $g$  is  $R_\delta^\rightarrow(g) := \min\{\text{cost}(\Pi) : \Pi \text{ solves } g \text{ to error } \delta\}$ .

If  $\Pi$  never uses  $R$ , it is deterministic. Minimizing over zero-error deterministic proto-

cols gives us the one-way deterministic communication complexity of  $g$ , denoted  $D^{\rightarrow}(g)$ .

**A Result on Random Graphs.** During the proof of our main lower bound (in Section 3.2.4), we shall need the following basic lemma on the maximum degree of a random graph.

**Lemma 3.2.3.** *Let  $G$  be a graph with  $M$  edges and  $n$  vertices, drawn uniformly at random. Define  $\Delta_G$  to be its maximum degree. Then for  $0 \leq \varepsilon \leq 1$ :*

$$\Pr \left[ \Delta_G \geq \frac{2M}{n}(1 + \varepsilon) \right] \leq 2n \exp \left( -\frac{\varepsilon^2}{3} \cdot \frac{2M}{n} \right). \quad (3.1)$$

*Proof.* Let  $G(n, m)$  be the uniform distribution over graphs with  $m$  edges and  $n$  vertices. Observe the monotonicity property that for all  $m \in \mathbb{N}$ ,  $\Pr_{G \sim G(n, m)}[\Delta_G \geq C] \leq \Pr_{G \sim G(n, m+1)}[\Delta_G \geq C]$ . Next, let  $H(n, p)$  be the distribution over graphs on  $n$  vertices in which each edge is included with probability  $p$ , independently of any others, and let  $e(G)$  be the number of edges of a given graph  $G$ . Then with  $p = M/\binom{n}{2}$ ,

$$\begin{aligned} \Pr_{G \sim G(n, M)}[\Delta_G \geq C] &= \Pr_{G \sim H(n, p)}[\Delta_G \geq C \mid e(G) = M] \leq \Pr_{G \sim H(n, p)}[\Delta_G \geq C \mid e(G) \geq M] \quad \triangleleft \text{by monotonicity} \\ &\leq \frac{\Pr_{G \sim H(n, p)}[\Delta_G \geq C]}{\Pr_{G \sim H(n, p)}[e(G) \geq M]} \leq 2 \Pr_{G \sim H(n, p)}[\Delta_G \geq C]. \end{aligned}$$

The last step follows from the well-known fact that the median of a binomial distribution equals its expectation when the latter is integral; hence  $\Pr_{G \sim H(n, p)}[e(G) \geq M] \geq 1/2$ .

Taking  $C = (2M/n)(1 + \varepsilon)$  and using a union bound and Chernoff's inequality,

$$\Pr_{G \sim H(n, p)} \left[ \Delta_G \geq \frac{2M}{n}(1 + \varepsilon) \right] \leq \sum_{x \in V(G)} \Pr_{G \sim H(n, p)} \left[ \deg_G(x) \geq \frac{2M}{n}(1 + \varepsilon) \right] \leq n \exp \left( -\frac{\varepsilon^2}{3} \cdot \frac{2M}{n} \right).$$

**Algorithmic Results From Prior Work.** Our adversarially robust graph coloring algorithms in Section 3.2.7 will use, as subroutines, some previously known standard streaming

algorithms for coloring. We summarize the key properties of these existing algorithms.

**Fact 3.2.1** (Restatement of [17], Result 2). *There is a randomized turnstile streaming algorithm for  $(\Delta + 1)$ -coloring a graph with max-degree  $\Delta$  in the oblivious adversary setting that uses  $\tilde{O}(n)$  bits of space and  $\tilde{O}(n)$  random bits. The failure probability can be made at most  $1/n^p$  for any large constant  $p$ .*  $\square$

In the adversarial model described above, we need to answer a query after each stream update. The algorithm mentioned in Fact 3.2.1 or other known algorithms using “about”  $\Delta$  colors (e.g., [43]) use at least  $\tilde{\Theta}(n)$  post-processing time in the worst case to answer a query. Hence, using such algorithms in the adaptive adversary setting might be inefficient. We observe, however, that at least for insert-only streams, there exists an algorithm that is efficient in terms of both space and time. This is obtained by combining the algorithms of [43] and [102] (see the discussion towards the end of Section 3.2.7 for details).

**Fact 3.2.2.** *In the oblivious adversary setting, there is a randomized streaming algorithm that receives a stream of edge insertions of a graph with max-degree  $\Delta$  and degeneracy  $\kappa$  and maintains a proper coloring of the graph using  $\kappa(1 + \varepsilon) \leq \Delta(1 + \varepsilon)$  colors,  $\tilde{O}(\varepsilon^{-2}n)$  space, and  $O(1)$  amortized update time. The failure probability can be made at most  $1/n^p$  for any large constant  $p$ .*  $\square$

### 3.2.3. Overview of Techniques

**Lower Bound Techniques.** As might be expected, our lower bounds are best formalized through communication complexity. Recall that a typical communication-to-streaming reduction for proving a one-pass streaming space lower bound works as follows. We set up a communication game for Alice and Bob to solve, using one message from Alice to Bob. Suppose that Alice and Bob have inputs  $x$  and  $y$  in this game. The players simulate a purported efficient streaming algorithm  $\mathcal{A}$  (for  $P$ , the problem of interest) by having Alice feed some tokens into  $\mathcal{A}$  based on  $x$ , communicating the resulting memory state of  $\mathcal{A}$  to

Bob, having Bob continue feeding tokens into  $\mathcal{A}$  based on  $y$ , and finally querying  $\mathcal{A}$  for an answer to  $P$ , based on which Bob can give a good output in the communication game. When this works, it follows that the space used by  $\mathcal{A}$  must be at least the one-way (and perhaps randomized) communication complexity of the game. Note, however, that this style of argument where it is possible to solve the game by querying the algorithm only once, is also applicable to an oblivious adversary setting. Therefore, it cannot prove a lower bound any higher than the standard streaming complexity of  $P$ .

The way to obtain stronger lower bounds by using the purported adversarial robustness of  $\mathcal{A}$  is to design communication protocols where Bob, after receiving Alice's message, proceeds to query  $\mathcal{A}$  repeatedly, feeding tokens into  $\mathcal{A}$  based on answers to such queries. In fact, in the communication games we shall use for our reductions, Bob will not have any input at all and the goal of the game will be for Bob to recover information about Alice's input, perhaps indirectly. It should be clear that the lower bound for the MIF problem, outlined in Section 4.2.1, can be formalized in this manner. For our main lower bound (Theorem 3.2.1), we use a communication game that can be seen as a souped-up version of MIF.

**The Subset-Avoidance Problem.** Recall the SUBSET-AVOIDANCE problem described in Section 3.2.1 and denote it  $\text{AVOID}(t, a, b)$ . To restate: Alice is given a set  $A \subseteq [t]$  of size  $a$  and must induce Bob to output a set  $B \subseteq [t]$  of size  $b$  such that  $A \cap B = \emptyset$ . The one-way communication complexity of this game can be lower bounded from first principles. Since each output of Bob is compatible with only  $\binom{t-b}{a}$  possible input sets of Alice, she cannot send the same message on more than that many inputs. Therefore, she must be able to send roughly  $\binom{t}{a} / \binom{t-b}{a}$  distinct messages for a protocol to succeed with high probability. The number of bits she must communicate in the worst case is roughly the logarithm of this ratio, which we show is  $\Omega(ab/t)$ . Interestingly, this lower bound is tight and can in fact be matched by a deterministic protocol, as shown in Lemma 3.2.5.

In the sequel, we shall need to consider a direct sum version of this problem that we call  $\text{AVOID}^k(t, a, b)$ , where Alice has a list of  $k$  subsets and Bob must produce his own list of subsets, with his  $i$ th avoiding the  $i$ th subset of Alice. We extend our lower bound argument to show that the one-way complexity of  $\text{AVOID}^k(t, a, b)$  is  $\Omega(kab/t)$ .

**Using Graph Coloring to Solve Subset-Avoidance.** To explain how we reduce the  $\text{AVOID}^k$  problem to graph coloring, we focus on a special case of Theorem 3.2.1 first. Suppose we have an adversarially robust  $(\Delta + 1)$ -coloring streaming algorithm  $\mathcal{A}$ . We describe a protocol for solving  $\text{AVOID}(t, a, b)$ . Let us set  $t = \binom{n}{2}$  to have the universe correspond to all possible edges of an  $n$ -vertex graph. Suppose Alice's set  $A$  has size  $a \approx n^2/8$ . We show that, given a set of  $n$  vertices, Alice can use public randomness to randomly map her elements to the set of vertex-pairs so that the corresponding edges induce a graph  $G$  that, w.h.p., has max-degree  $\Delta \approx n/4$ . Alice proceeds to feed the edges of  $G$  into  $\mathcal{A}$  and then sends Bob the state of  $\mathcal{A}$ .

Bob now queries  $\mathcal{A}$  to obtain a  $(\Delta + 1)$ -coloring of  $G$ . Then, he pairs up like-colored vertices to obtain a maximal pairing. Observe that he can pair up all but at most one vertex from each color class. Thus, he obtains at least  $(n - \Delta - 1)/2$  such pairs. Since each pair is monochromatic, they don't share an edge, and hence, Bob has retrieved  $(n - \Delta - 1)/2$  missing edges that correspond to elements absent in Alice's set. Since Alice used public randomness for the mapping, Bob knows exactly which elements these are. He now forms a matching with these pairs and inserts the edges to  $\mathcal{A}$ . Once again, he queries  $\mathcal{A}$  to find a coloring of the modified graph. Observe that the matching can increase the max-degree of the original graph by at most 1. Therefore, this new coloring uses at most  $\Delta + 2$  colors. Thus, Bob would retrieve at least  $(n - \Delta - 2)/2$  new missing edges. He again adds to the graph the matching formed by those edges and queries  $\mathcal{A}$ . It is crucial to note here that he can repeatedly do this and expect  $\mathcal{A}$  to output a correct coloring because of its adversarial robustness. Bob stops once the max-degree reaches  $n - 1$ , since now the algorithm can

color each vertex with a distinct color, preventing him from finding a missing edge.

Summing up the sizes of all the matchings added by Bob, we see that he has found  $\Theta((n - \Delta)^2)$  elements missing from Alice's set. Since  $\Delta \approx n/4$ , this is  $\Theta(n^2)$ . Thus, Alice and Bob have solved the  $\text{AVOID}(t, a, b)$  problem where  $t = \binom{n}{2}$  and  $a, b = \Theta(n^2)$ . As outlined above, this requires  $\Omega(ab/t) = \Omega(n^2)$  communication. Hence,  $\mathcal{A}$  must use at least  $\Omega(n^2) = \Omega(n\Delta)$  space.

With some further work, we can generalize the above argument to work for any value of  $\Delta$  with  $1 \leq \Delta \leq n/2$ . For this generalization, we use the communication complexity of  $\text{AVOID}^k(t, a, b)$  for suitable parameter settings. With more rigorous analysis, we can further generalize the result to apply not only to  $(\Delta + 1)$ -coloring algorithms but to any  $f(\Delta)$ -coloring algorithm. That is, we can prove Theorem 3.2.6.

**Upper Bound Techniques.** It is useful to outline our algorithms in an order different from the presentation in Section 3.2.5.

**A Sketch-Switching-Based  $O(\Delta^2)$ -Coloring.** The main challenge in designing an adversarially robust coloring algorithm is that the adversary can compel the algorithm to change its output at every point in the stream: he queries the algorithm, examines the returned coloring, and inserts an edge between two vertices of the same color. Indeed, the sketch switching framework of [40] shows that for *function estimation*, one can get around this power of the adversary as follows. Start with a basic (i.e., oblivious-adversary) sketch for the problem at hand. Then, to deal with an adaptive adversary, run multiple independent basic sketches in parallel, changing outputs only when forced to because the underlying function has changed significantly. More precisely, maintain  $\lambda$  independent parallel sketches where  $\lambda$  is the *flip number*, defined as the maximum number of times the function value can change by the desired approximation factor over the course of the stream. Keep track of which sketch is currently being used to report outputs to the adversary. Upon being queried, re-use the most recently given output unless forced to change, in which case

discard the current sketch and switch to the next in the list of  $\lambda$  sketches. Notice that this keeps the adversary oblivious to the randomness being used to compute future outputs: as soon as our output reveals any information about the current sketch, we discard it and never use it again to process a stream element.

This way of switching to a new sketch only when forced to ensures that  $\lambda$  sketches suffice, which is great for function estimation. However, since a graph coloring output can be forced to change at every point in a stream of length  $m$ , naively implementing this idea would require  $m$  parallel sketches, incurring a factor of  $m$  in space. We have to be more sophisticated. We combine the above idea with a chunking technique so as to reduce the number of times we need to switch sketches.

Suppose we split the  $m$ -length stream into  $k$  chunks, each of size  $m/k$ . We initialize  $k$  parallel sketches of a standard streaming  $(\Delta + 1)$ -coloring algorithm  $\mathcal{C}$  to be used one at a time as each chunk ends. We store (buffer) an entire chunk explicitly and when we reach its end, we say we have reached a “checkpoint,” use a fresh copy of  $\mathcal{C}$  to compute a  $(\Delta + 1)$ -coloring of the entire graph at that point, delete the chunk from our memory, and move on to store the next chunk. When a query arrives, we deterministically compute a  $(\Delta + 1)$ -coloring of the partial chunk in our buffer and “combine” it with the coloring we computed at the last checkpoint. The combination uses at most  $(\Delta + 1)^2 = O(\Delta^2)$  colors. Since a single copy of  $\mathcal{C}$  takes  $\tilde{O}(n)$  space, the total space used by the sketches is  $\tilde{O}(nk)$ . Buffering a chunk uses an additional  $\tilde{O}(m/k)$  space. Setting  $k$  to be  $\sqrt{m/n}$ , we get the total space usage to be  $\tilde{O}(\sqrt{mn}) = \tilde{O}(n\sqrt{\Delta})$ , since  $m = O(n\Delta)$ .

Handling edge deletions is more delicate. This is because we can no longer express the current graph as a union of  $G_1$  (the graph up to the most recent checkpoint) and  $G_2$  (the buffered subgraph) as above. A chunk may now contain an update that deletes an edge which was inserted before the checkpoint, and hence, is not in store. Observe, however, that deleting an edge doesn’t violate the validity of a coloring. Hence, if we ignore these

edge deletions, the only worry is that they might substantially reduce the maximum degree  $\Delta$  causing us to use many more colors than desired. Now, note that if we have a  $(\Delta_1 + 1)$ -coloring at the checkpoint, then as long as the current maximum degree  $\Delta$  remains above  $\Delta_1/2$ , we have a  $2\Delta$ -coloring in store. Hence, combining that with a  $(\Delta + 1)$ -coloring of the current chunk gives an  $O(\Delta^2)$ -coloring. Furthermore, we can keep track of the maximum degree of the graph using only  $\tilde{O}(n)$  space and detect the points where it falls below half of what it was at the last checkpoint. We declare each such point as a new “ad hoc checkpoint,” i.e., use a fresh sketch to compute a  $(\Delta + 1)$ -coloring there. Since the max-degree can decrease by a factor of 2 at most  $\log n$  times, we show that it suffices to have only  $\log n$  times more parallel sketches initialized at the beginning of the stream. This incurs only an  $O(\log n)$ -factor overhead in space. We discuss the algorithm and its analysis in detail in Algorithm 4 and Lemma 3.2.15 respectively.

To generalize the above to an  $O(\Delta^k)$ -coloring in  $\tilde{O}(n\Delta^{1/k})$  space, we use recursion in a manner reminiscent of streaming coreset construction algorithms. Split the stream into  $\Delta^{1/k}$  chunks, each of size  $n\Delta^{1-1/k}$ . Now, instead of storing a chunk entirely and coloring it deterministically, we can recursively color it with  $\Delta^{k-1}$  colors in  $O(n\Delta^{1/k})$  space and combine the coloring with the  $(\Delta + 1)$ -coloring at the last checkpoint. The recursion makes the analysis of this algorithm even more delicate, and careful work is needed to argue the space usage and to properly handle deletions in the turnstile setting. The details appear in Theorem 3.2.16.

**A Palette-Sparsification-Based  $O(\Delta^3)$ -Coloring.** This algorithm uses a different approach to the problem of the adversary forcing color changes. It ensures that, every time an edge is added, one of its endpoints is randomly recolored, where the color is drawn uniformly from a set  $C \setminus K$  of colors, where  $C$  is determined by the degree of the endpoint, and  $K$  is the set of colors currently held by neighboring vertices. Let  $R_v$  denote the random string that drives this color-choosing process at vertex  $v$ . When the adversary inserts

an edge  $\{u, v\}$ , the algorithm uses  $R_u$  and  $R_v$  to determine whether this edge could with significant probability end up with the same vertex color on both ends in the future. If so, the algorithm stores the edge; if not, it can be ignored entirely. It will turn out that when the number of colors is set to establish an  $O(\Delta^3)$ -coloring, only an  $\tilde{O}(1/\Delta)$  fraction of edges need to be stored, so the algorithm only needs to store  $\tilde{O}(n)$  bits of data related to the input. The proof of this storage bound has to contend with an adaptive adversary. We do so by first arguing that despite this adaptivity, the adversary cannot cause the algorithm to use more storage than the worst oblivious adversary could have. We can then complete the proof along traditional lines, using concentration bounds. The details appear in Algorithm 3 and Theorem 3.2.12.

There is a technical caveat here. The random string  $R_v$  used at each vertex  $v$  is about  $\tilde{O}(\Delta)$  bits long. Thus, the algorithm can only be called semi-streaming if we agree that these  $\tilde{O}(n\Delta)$  random bits do not count towards the storage cost. In the standard streaming setting, this “randomness cost” is not a concern, for we can use the standard technique of invoking Nisan’s space-bounded pseudorandom generator [146] to argue that the necessary bits can be generated on the fly and never stored. Unfortunately, it is not clear that this transformation preserves adversarial robustness. Despite this caveat, the algorithmic result is interesting as a contrast to our lower bounds, because the lower bounds do apply even in a model where random bits are free, and only actually computed input-dependent bits count towards the space complexity.

### 3.2.4. Hardness of Adversarially Robust Graph Coloring

---

In this section, we prove our first major result, showing that graph coloring is significantly harder when working against an adaptive adversary than it is in the standard setting of an oblivious adversary. We carry out the proof plan outlined in Section 3.2.3, first describing and analyzing our novel communication game of SUBSET-AVOIDANCE (henceforth, AVOID) and then reducing the AVOID problem to robust coloring.

**The Subset Avoidance Problem.** Let  $\text{AVOID}(t, a, b)$  denote the following one-way communication game.

- Alice is given  $S \subseteq [t]$  with  $|S| = a$ ;
- Bob must produce  $T \subseteq [t]$  with  $|T| = b$  for which  $T$  is disjoint from  $S$ .

Let  $\text{AVOID}^k(t, a, b)$  be the problem of simultaneously solving  $k$  instances of  $\text{AVOID}(t, a, b)$ .

**Lemma 3.2.4.** *The public-coin  $\delta$ -error communication complexity of  $\text{AVOID}^k(t, a, b)$  is bounded thus:*

$$R_{\delta}^{\rightarrow}(\text{AVOID}^k(t, a, b)) \geq \log(1 - \delta) + k \log \left( \binom{t}{a} / \binom{t-b}{a} \right) \quad (3.2)$$

$$\geq \log(1 - \delta) + kab / (t \ln 2). \quad (3.3)$$

*Proof.* Let  $\Pi$  be a  $\delta$ -error protocol for  $\text{AVOID}^k(t, a, b)$  and let  $d = \text{cost}(\Pi)$ , as defined in Section 3.2.2. Since, for each input  $(S_1, \dots, S_k) \in \binom{[t]}{a}^k$ , the error probability of  $\Pi$  on that input is at most  $\delta$ , there must exist a fixing of the random coins of  $\Pi$  so that the resulting deterministic protocol  $\Pi'$  is correct on all inputs in a set

$$\mathcal{C} \subseteq \binom{[t]}{a}^k, \quad \text{with } |\mathcal{C}| \geq (1 - \delta) \binom{t}{a}^k.$$

The protocol  $\Pi'$  is equivalent to a function  $\phi: \mathcal{C} \rightarrow \binom{[t]}{b}^k$  where

- the range size  $|\text{Im}(\phi)| \leq 2^d$ , because  $\text{cost}(\Pi) \leq d$ , and
- for each  $(S_1, \dots, S_k) \in \mathcal{C}$ , the tuple  $(T_1, \dots, T_k) := \phi((S_1, \dots, S_k))$  is a correct output for Bob, i.e.,  $S_i \cap T_i = \emptyset$  for each  $i$ .

For any fixed  $(T_1, \dots, T_k) \in \binom{[t]}{b}^k$ , the set of all  $(S_1, \dots, S_k) \in \binom{[t]}{a}^k$  for which each coordinate  $S_i$  is disjoint from the corresponding  $T_i$  is precisely the set  $\binom{[t] \setminus T_1}{S_1} \times \dots \times \binom{[t] \setminus T_k}{S_k}$ .

The cardinality of this set is exactly  $\binom{t-b}{a}^k$ . Thus, for any subset  $\mathcal{D}$  of  $\binom{[t]}{b}^k$ , it holds that  $|\mathcal{C} \cap \phi^{-1}(\mathcal{D})| \leq \binom{t-b}{a}^k |\mathcal{D}|$ . Consequently,

$$(1 - \delta) \binom{t}{a}^k \leq |\mathcal{C}| = |\phi^{-1}(\text{Im}(\phi))| \leq \binom{t-b}{a}^k |\text{Im}(\phi)| \leq \binom{t-b}{a}^k 2^d,$$

which, on rearrangement, gives eq. (3.2).

To obtain eq. (3.3), we note that

$$\begin{aligned} \binom{t}{a} / \binom{t-b}{a} &= \frac{t!a!(t-a-b)!}{(t-a)!a!(t-b)!} = \frac{t \cdot (t-1) \cdots (t-a+1)}{(t-b) \cdot (t-b-1) \cdots (t-a-b+1)} \\ &\geq \left(\frac{t}{t-b}\right)^a = \left(\frac{1}{1-b/t}\right)^a > e^{ab/t}, \end{aligned} \quad (3.4)$$

which implies

$$\log(1 - \delta) + k \log \left( \binom{t}{a} / \binom{t-b}{a} \right) \geq \log(1 - \delta) + kab/(t \ln 2). \quad \square$$

Since our data streaming lower bounds are based on the  $\text{AVOID}^k$  problem, it is important to verify that we are not analyzing its communication complexity too loosely. To this end, we prove the following result, which says that the lower bound in Lemma 3.2.4 is close to being tight. In fact, a nearly matching upper bound can be obtained deterministically.

**Lemma 3.2.5.** *For any  $t \in \mathbb{N}$ ,  $0 < a + b \leq t$ , the deterministic complexity of  $\text{AVOID}(t, a, b)$  is bounded thus:*

$$D^\rightarrow(\text{AVOID}(t, a, b)) \leq \log \left( \binom{t}{a} / \binom{t-b}{a} \right) + \log \left( \ln \binom{t}{a} \right) + 2. \quad (3.5)$$

*Proof.* We claim there exists an ordered collection  $\mathcal{R}$  of  $z := \lceil ((\binom{t}{a}) / \binom{t-b}{a}) \ln \binom{t}{a} \rceil$  subsets of  $[t]$  of size  $b$ , with the property that for each  $S \in \binom{[t]}{a}$ , there exists a set  $T$  in  $\mathcal{R}$  which is disjoint from  $S$ . In this case, Alice's protocol is, given a set  $S \in \binom{[t]}{a}$ , to send the index  $j$  of

the first set  $T$  in  $\mathcal{R}$  which is disjoint from  $S$ ; Bob in turn returns the  $j$ th element of  $\mathcal{R}$ . The number of bits needed to communicate such an index is at most  $\lceil \log z \rceil$ , implying eq. (3.5).

We prove the existence of such an  $\mathcal{R}$  by the probabilistic method. Pick a subset  $\mathcal{Q} \subseteq \binom{[t]}{b}$  of size  $z$  uniformly at random. For any  $S \in \binom{[t]}{a}$ , define  $\mathcal{O}_S$  to be the set of subsets in  $\binom{[t]}{b}$  which are disjoint from  $S$ ; observe that  $|\mathcal{O}_S| = \binom{t-a}{b}$ . Then  $\mathcal{Q}$  has the desired property if for all  $S \in \binom{[t]}{a}$ , it overlaps with  $\mathcal{O}_S$ . As

$$\begin{aligned}
 \Pr \left[ \exists S \in \binom{[t]}{a} : \mathcal{Q} \cap \mathcal{O}_S = \emptyset \right] &\leq \sum_{S \in \binom{[t]}{a}} \Pr [\mathcal{Q} \cap \mathcal{O}_S = \emptyset] && \triangleleft \text{by union bound} \\
 &= \sum_{S \in \binom{[t]}{a}} \Pr \left[ \mathcal{Q} \in \binom{\binom{[t]}{b} \setminus \mathcal{O}_S}{z} \right] \\
 &= \sum_{S \in \binom{[t]}{a}} \left( \left( \binom{t}{b} - \binom{t-a}{b} \right) / \binom{t}{b} \right)^z \\
 &< \binom{t}{a} \exp \left( -z \binom{t-a}{b} / \binom{t}{b} \right) && \triangleleft \text{by eq. (3.4)} \\
 &= \binom{t}{a} \exp \left( -z \binom{t-b}{a} / \binom{t}{a} \right),
 \end{aligned}$$

setting  $z = \lceil ((\binom{t}{a} / \binom{t-b}{a}) \ln \binom{t}{a}) \rceil$  ensures the random set  $\mathcal{Q}$  fails to have the desired property with probability strictly less than 1. Let  $\mathcal{R}$  be a realization of  $\mathcal{Q}$  that does have the property. □

**Reducing Multiple Subset Avoidance to Graph Coloring.** Having introduced and analyzed the AVOID communication game, we are now ready to prove our main lower bound result, on the hardness of adversarially robust graph coloring.

**Theorem 3.2.6** (Main lower bound). *Let  $L, n, K$  be integers with  $2K \leq n$ , and  $L+1 \leq K$ , and  $L \geq 12 \ln(4n)$ .*

*Assume there is an adversarially robust coloring algorithm  $\mathcal{A}$  for insert-only streams of  $n$ -vertex graphs which works as long as the input graph has maximum degree  $\leq L$ , and*

*maintains a coloring with  $\leq K$  colors so that all colorings are correct with probability  $\geq 1/4$ . Then  $\mathcal{A}$  requires at least  $C$  bits of space, where*

$$C \geq \frac{1}{40 \ln 2} \cdot \frac{nL^2}{K} - 3.$$

*Proof.* Given an algorithm  $\mathcal{A}$  as specified, we can construct a public-coin protocol to solve the communication problem  $\text{AVOID}^{\lfloor n/(2K) \rfloor}(\binom{2K}{2}, \lfloor LK/4 \rfloor, \lfloor L/2 \rfloor \lceil K/2 \rceil)$  using exactly as much communication as  $\mathcal{A}$  requires storage space. The protocol for the more basic problem  $\text{AVOID}(\binom{2K}{2}, \lfloor LK/4 \rfloor, \lfloor L/2 \rfloor \lceil K/2 \rceil)$  is described in Algorithm 2.

---

**Algorithm 2** Protocol for  $\text{AVOID}(\binom{2K}{2}, \lfloor LK/4 \rfloor, \lfloor L/2 \rfloor \lceil K/2 \rceil)$

---

**Require:** Algorithm  $\mathcal{A}$  that colors graphs up to maximum degree  $L$ , always using  $\leq K$  colors

- 1:  $R \leftarrow$  publicly random bits to be used by  $\mathcal{A}$
- 2:  $\pi \leftarrow$  publicly random permutation of  $\{1, \dots, \binom{2K}{2}\}$ , drawn uniformly
- 3:  $e_1, \dots, e_{\binom{2K}{2}} \leftarrow$  an enumeration of the edges of the complete graph on  $2K$  vertices
- 4: **function** ALICE( $S$ ):
- 5:      $Z \leftarrow \mathcal{A}::\text{INIT}(R)$ , the initial state of  $\mathcal{A}$
- 6:     **for**  $i$  from 1 to  $\binom{2K}{2}$  **do**
- 7:         **if**  $\pi_i \in S$  **then**
- 8:              $Z \leftarrow \mathcal{A}::\text{INSERT}(Z, R, e_i)$
- 9:     **return**  $Z$
- 10: **function** BOB( $Z$ ):
- 11:      $J \leftarrow$  empty list
- 12:     **for**  $i$  from 1 to  $\lfloor L/2 \rfloor$  **do**
- 13:          $\text{CLR} \leftarrow \mathcal{A}::\text{QUERY}(Z, R)$
- 14:          $M \leftarrow$  maximal pairing of like-colored vertices, according to CLR
- 15:         **for** each pair  $\{u, v\} \in M$  **do**
- 16:              $Z \leftarrow \mathcal{A}::\text{INSERT}(Z, R, \{u, v\})$        $\triangleright M$  is turned into a matching and inserted
- 17:      $J \leftarrow J \cup M$
- 18:     **if**  $\text{length}(J) \leq \lfloor L/2 \rfloor \lceil K/2 \rceil$  **then**
- 19:         **return** fail
- 20:     **else**
- 21:          $T \leftarrow \{\pi_i : e_i \in \text{first } \lfloor L/2 \rfloor \lceil K/2 \rceil \text{ edges of } J\}$
- 22:     **return**  $T$

---

To use  $\mathcal{A}$  to solve  $s := \lfloor n/2K \rfloor$  instances of AVOID, we pick  $s$  disjoint subsets  $V_1, \dots, V_s$  of the vertex set  $[n]$ , each of size  $2K$ . A streaming coloring algorithm on the vertex set  $[2K]$  with degree limit  $L$  and using at most  $K$  colors can be implemented by relabeling the vertices in  $[2K]$  to the vertices in some set  $V_i$  and using  $\mathcal{A}$ . This can be done  $s$  times in parallel, as the sets  $(V_i)_{i=1}^s$  are disjoint. Note that a coloring of the entire graph on vertex set  $[n]$  using  $\leq K$  colors is also a  $K$ -coloring of the  $s$  subgraphs supported on  $V_1, \dots, V_s$ . To minimize the number of color queries made, Algorithm 2 can be implemented by alternating between adding elements from the matching  $M$  in each instance (for Line 16), and making single color queries to the  $n$ -vertex graph (for Line 13).

The guarantee that  $\mathcal{A}$  uses fewer than  $K$  colors depends on the input graph stream having maximum degree at most  $L$ . In Bob's part of the protocol, adding a matching to the graph only increases the maximum degree of the graph represented by  $Z$  by at most one; since he does this  $\lfloor L/2 \rfloor$  times, in order for the maximum degree of the graph represented by  $Z$  to remain at most  $L$ , we would like the random graph Alice inserts into the algorithm to have maximum degree  $\leq L/2 \leq L - \lfloor L/2 \rfloor$ . By Lemma 3.2.3, the probability that, given some  $i$ , this random graph on  $V_i$  has maximum degree  $\Delta_i \geq L/2$  is

$$\Pr \left[ \Delta_i \geq \frac{L}{4}(1+1) \right] \leq 4K e^{-L/12}.$$

Taking a union bound over all  $s$  graphs, we find that

$$\Pr \left[ \max_{i \in [s]} \Delta_i \geq L/2 \right] \leq 4K \left\lfloor \frac{n}{2K} \right\rfloor e^{-L/12} \leq 2n e^{-L/12}.$$

We can ensure that this happens with probability at most  $1/2$  by requiring  $L \geq 12 \ln(4n)$ .

If all the random graphs produced by Alice have maximum degree  $\leq L/2$ , and the  $\lfloor L/2 \rfloor$  colorings requested by the protocol are all correct, then we will show that Bob's part of the protocol recovers at least  $\lfloor L/2 \rfloor \lceil K/2 \rceil$  edges for each instance. Since the algorithm

$\mathcal{A}$ 's random bits  $R$  and permutation random bits  $\pi$  are independent, the probability that the the maximum degree is low and the algorithm gives correct colorings on graphs of maximum degree at most  $L$  is  $\geq (1/2) \cdot (1/4) = 1/8$ .

The list of edges that Bob inserts (Line 16) are fixed functions of the query output of  $\mathcal{A}$  on its state  $Z$  and random bits  $R$ . None of the edges can already have been inserted by Alice or Bob, since each edge connects two vertices which have the same color. Because these edges only depend on the query output of  $\mathcal{A}$ , conditioned on this query output they are independent of  $Z$  and  $R$ . This ensures that  $\mathcal{A}$ 's correctness guarantee against an adversary applies here, and thus the colorings reported on Line 13 are correct.

Assuming all queries succeed, and the initial graph that Alice added has maximum degree  $\leq L/2$ , for each  $i \in \llbracket L/2 \rrbracket$ , the coloring produced will have at most  $K$  colors. Let  $B$  be the set of vertices covered by the matching  $M$ , so that  $[2K] \setminus B$  are the unmatched vertices. Since no pair of unmatched vertices can have the same color,  $|[2K] \setminus B| \leq K$ . This implies  $|B| \geq K$ , and since  $|M| = |B|/2$  is an integer, we have  $|M| \geq \lceil K/2 \rceil$ . Thus each **for** loop iteration will add at least  $\lceil K/2 \rceil$  new edges to  $J$ . The final value of the list  $J$  will contain at least  $\lfloor L/2 \rfloor \lceil K/2 \rceil$  edges that were not added by Alice; Line 21 converts the first  $\lfloor L/2 \rfloor \lceil K/2 \rceil$  of these to elements of  $\{1, \dots, \binom{2K}{2}\}$  not in the set  $S$  given to Alice.

Finally, by applying Lemma 3.2.4, we find that the communication  $C$  needed to solve  $s$  independent copies of  $\text{AVOID}(\binom{2K}{2}, \lfloor LK/4 \rfloor, \lfloor L/2 \rfloor \lceil K/2 \rceil)$  with failure probability  $\leq 7/8$  satisfies

$$\begin{aligned} C &\geq \log \left( 1 - \frac{7}{8} \right) + \left\lfloor \frac{n}{2K} \right\rfloor \frac{\lfloor LK/4 \rfloor \cdot \lfloor L/2 \rfloor \lceil K/2 \rceil}{\binom{2K}{2} \ln 2} \\ &\geq \frac{n}{4K} \frac{L^2 K^2 / 20}{\frac{1}{2} (2K)^2 \ln 2} - 3 \geq \frac{n L^2}{40K \ln 2} - 3, \end{aligned}$$

where we used  $K > L \geq 12 \ln(4n) \geq 12 \ln 4$  to conclude  $\lfloor LK/4 \rfloor \lfloor L/2 \rfloor \lceil K/2 \rceil \geq (LK)^2/20$ .  $\square$

Applying the above Theorem 3.2.6 with “ $K = f(L)$ ,” we immediately obtain the following corollary, which highlights certain parameter settings that are particularly instructive.

**Corollary 3.2.7.** *Let  $f$  be a monotonically increasing function, and  $L$  an integer for which  $L = \Omega(\log n)$  and  $f(L) \leq n/2$ . Let  $\mathcal{A}$  be a coloring algorithm which works for graphs of maximum degree up to  $L$ ; which at any point in time uses  $\leq f(\Delta)$  colors, where  $\Delta$  is the current graph’s maximum degree; and which has total failure probability  $\leq 3/4$  against an adaptive adversary. Then the number of bits  $S$  of space used by  $\mathcal{A}$  is lower-bounded as  $S = \Omega(nL^2/f(L))$ . In particular:*

- *If  $f(\Delta) = \Delta + 1$ —or, more generally,  $f(\Delta) = O(\Delta)$ —then  $S = \Omega(nL)$  space is needed.*
- *To ensure  $S = \tilde{O}(n)$  space,  $f(\Delta) = \tilde{\Omega}(\Delta^2)$  is needed.*
- *If  $f(L) = \Theta(n)$ , then  $S = \Omega(L^2)$ .* □

### 3.2.5. Upper Bounds: Adversarially Robust Coloring Algorithms

We now turn to positive results. We show how to maintain a  $\text{poly}(\Delta)$ -coloring of a graph in an adversarially robust fashion. We design two broad classes of algorithms. The first, described in Section 3.2.6, is based on palette sparsification as in [8, 17], with suitable enhancements to ensure robustness. The resulting algorithm maintains an  $O(\Delta^3)$ -coloring and uses  $\tilde{O}(n)$  bits of working memory. As noted in Section 3.2.3, the algorithm comes with the caveat that it requires a large pool of random bits: up to  $\tilde{O}(n\Delta)$  of them. As also noted there, it makes sense to treat this randomness cost as separate from the space cost.

The second class of algorithms, described in Section 3.2.7, is built on top of the sketch switching technique of [40], suitably modified to handle non-real-valued outputs. This time, the amount of randomness used is small enough that we can afford to store all random bits in working memory. These algorithms can be enhanced to handle strict graph turnstile

streams as described in Section 3.2.2. For any such turnstile stream of length at most  $m$ , we maintain an  $O(\Delta^2)$ -coloring using  $\tilde{O}(\sqrt{nm})$  space. More generally, we maintain an  $O(\Delta^k)$ -coloring in  $O(n^{1-1/k}m^{1/k})$  space for any  $k \in \mathbb{N}$ . In particular, for insert-only streams, this implies an  $O(\Delta^k)$ -coloring in  $O(n\Delta^{1/k})$  space.

### 3.2.6. An Algorithm Based on Palette Sparsification

We proceed to describe our palette-sparsification-based algorithm. It maintains a  $3\Delta^3$ -coloring of the input graph  $G$ , where  $\Delta$  is the evolving maximum degree of the input graph  $G$ . With high probability, it will store only  $O(n(\log n)^4) = \tilde{O}(n)$  bits of information about  $G$ ; an easy modification ensures that this bound is always maintained by having the algorithm abort if it is about to overshoot the bound.

The algorithm does need a large number of random bits—up to  $O(nL(\log n)^2)$  of them—where  $L$  is the maximum degree of the graph at the end of the stream or an upper bound on the same. Due to the way the algorithm looks ahead at future random bits,  $L$  must be known in advance.

The algorithm uses these available random bits to pick, for each vertex,  $L$  lists of random color palettes, one at each of  $L$  “levels.” The level- $i$  list at vertex  $x$  is called  $P_x^i$  and consists of  $4 \log n$  colors picked uniformly at random with replacement from the set  $[2i^2]$ . The algorithm tracks each vertex’s degree. Whenever a vertex  $x$  is recolored, its new color is always of the form  $(d, p)$ , where  $d = \deg(x)$  and  $p \in P_x^d$ . Thus, when the maximum degree in  $G$  is  $\Delta$ , the only colors that have been used are the initial default  $(0, 0)$  and colors from  $\bigcup_{i=1}^{\Delta} \{i\} \times [2i^2]$ . The total number of colors is therefore at most  $1 + \sum_{i=1}^{\Delta} 2i^2 \leq 3\Delta^3$ .

The precise algorithm is given in Algorithm 3.

**Lemma 3.2.8** (Bounding the failure probability). *When an edge is added, recoloring one of its vertices succeeds with probability  $\geq 1 - 1/n^4$ , regardless of the past history of the algorithm.*

---

**Algorithm 3** Adversarially robust  $3\Delta^3$ -coloring algorithm, assuming  $0 < \Delta \leq L$ 


---

**Input:** Stream of edges of a graph  $G = (V, E)$ , with maximum degree always  $\leq L$ .

**Random bits:**

- 1: **for** each vertex  $x \in [n]$  **do**
- 2:     **for** each  $i \in [L]$  **do**
- 3:          $P_x^i \leftarrow$  list of  $4 \log n$  colors sampled u.a.r. with replacement from  $[2i^2]$

**Initialize:**

- 4: **for** each vertex  $x \in [n]$  **do**
- 5:      $\text{DEG}(x) \leftarrow 0$  ▷ tracks degree of  $x$
- 6:      $\text{CLR}(x) \leftarrow (0, 0)$  ▷ maintains color of  $x$ ; in general  $\in \bigcup_{i=1}^L \{i\} \times [2i^2]$
- 7:  $A \leftarrow$  empty list of edges

**Process(edge  $\{u, v\}$ ):**

- 8:  $\text{DEG}(u), \text{DEG}(v) \leftarrow \text{DEG}(u) + 1, \text{DEG}(v) + 1$  ▷ maintain vertex degrees
- 9:  $k \leftarrow \max\{\text{DEG}(u), \text{DEG}(v)\}$
- 10: **for**  $i$  from  $k$  to  $L$  **do** ▷ store edges that might be needed in the future
- 11:     **if**  $P_u^i$  and  $P_v^i$  overlap **then**
- 12:          $A \leftarrow A \cup \{\{u, v\}\}$
- 13:  $\text{USED} \leftarrow \{\text{CLR}(w) : \{u, w\} \in A\}$  ▷ prepare to recolor vertex  $u$ : collect colors of neighbors
- 14: **for**  $j$  from 1 to  $4 \log n$  **do**
- 15:      $c \leftarrow (\text{DEG}(u), P_u^{\text{DEG}(u)}[j])$  ▷ try the next color in the random list
- 16:     **if**  $c \notin \text{USED}$  **then**
- 17:          $\text{CLR}(u) \leftarrow c$ ; **return**
- 18: **abort** ▷ failed to find a color

**Query( )::**

- 19: **return** the vector  $\text{CLR}$
- 

*Proof.* The color for the endpoint  $u$  is chosen and assigned in Lines 13 through 17. Let  $d$  be the value of  $\text{DEG}(u)$  at that point. First, we observe that because the list  $P_u^d$  of colors to try was drawn independently of all other lists, and has never been used before by the algorithm, it is necessarily independent of the rest of the algorithm state.

A given color  $(d, P_u^d[j])$  is only invalid if there exists some other vertex  $w$  which has precisely this color. If this were the case, then the set  $\text{USED}$  would contain that color, because  $\text{USED}$  contains all colors on vertices  $w$  with  $\text{DEG}(w) = d$  and whose list of potential

colors  $P_w^d$  overlaps with  $P_u^d$ . Thus, the algorithm will detect any invalid colors in Line 16.

The probability that the algorithm fails to find a valid color is:

$$\Pr[P_u^d \subseteq \text{USED}] = \prod_{j=1}^{4 \log n} \Pr[P_u^d[j] \in \text{USED}] = \prod_{j=1}^{4 \log n} \frac{|\text{USED}|}{2d^2} \leq \frac{1}{2^{4 \log n}} = \frac{1}{n^4},$$

where the inequality uses the fact that  $|\text{USED}| \leq \text{DEG}(u) = d$ .  $\square$

Taking a union bound over the at most  $nL/2$  endpoints modified, we find that the total probability of a recoloring failure in the algorithm is, by Lemma 3.2.8, at most  $(1/n^4) \cdot nL/2 \leq 1/n^2$ .

The rest of this section is dedicated to analyzing the space cost of Algorithm 3. In general, an adaptive adversary could try to construct a bad sequence of updates that causes the algorithm to store too many edges. The next two lemmas argue that for Algorithm 3, the adversary is unable to use his adaptivity for this purpose: he can do no worse than the worst oblivious adversary. Subsequently, Lemma 3.2.11 shows that Algorithm 3 does well in terms of space cost against an oblivious adversary, which completes the analysis.

**Lemma 3.2.9.** *Let  $\tau = (e_1, \chi_1, e_2, \chi_2, \dots, \chi_{i-1}, e_i)$  be the transcript of the edges  $(e_1, \dots, e_i)$  that an adversary provides to an implementation of Algorithm 3, and of the colorings  $(\chi_1, \dots, \chi_{i-1})$  produced by querying after each of the first  $(i-1)$  edges was added. Let  $\sigma = (e_{i+1}, \dots, e_j)$  be an arbitrary sequence of edges not in  $\bigcup_{h=1}^i e_h$ , and let  $\gamma$  be a subsequence of  $\sigma$ . Conditioned on  $\tau$ , the next coloring  $\chi_i$  returned is independent of the event that when the next edges in the input stream are  $\sigma$ , the algorithm will store  $\gamma$  in its list  $A$ .*

*Proof.* Let  $G = \bigcup_{j=1}^i e_j$  be the graph containing all edges up to  $e_i$ , and let  $e_i = \{u, v\}$ , so that  $u$  is the vertex recolored in Lines 13 through 17. Let  $\deg_G(x)$  be the degree of vertex  $x$  in  $G$ . We can partition the array  $[n] \times [L]$  of indices for random color lists  $(P_x^i)_{(x,i) \in [n] \times [L]}$

used by Algorithm 3 into three groups, defined as follows:

$$\mathcal{Q}_> = \{(x, i) \in [n] \times [L] : i \geq \deg_G(x) + 1\}$$

$$\mathcal{Q}_= = \{(u, \deg_G(u))\}$$

$$\mathcal{Q}_< = \{(x, i) \in [n] \times [L] : i \leq \deg_G(x)\} \setminus \mathcal{Q}_=.$$

The next coloring  $\chi_i$  returned by the algorithm depends only on  $u$ ,  $G$ ,  $\chi_{i-1}$ , and the random color list  $P_u^{\deg_G(u)}$ . On the other hand, the past colorings  $(\chi_1, \dots, \chi_{i-1})$  returned by the algorithm depend only on  $(e_1, \dots, e_{i-1})$  and the color lists indexed by  $\mathcal{Q}_<$ . Finally, whether an edge  $\{a, b\}$  is stored in the set  $A$  in the future depends only on the edges added up to that time and some of the color lists from  $\mathcal{Q}_>$ , because (per Lines 9 to 12) only color lists  $P_a^i$  and  $P_b^i$  with  $i \geq \max(\text{DEG}(a), \text{DEG}(b))$  are considered. (Note that at the time the new edge is processed,  $\text{DEG}(a)$  and  $\text{DEG}(b)$  will both be larger than  $\deg_G(a)$  and  $\deg_G(b)$  because Line 8 will have increased the vertex degrees.) Also observe that the edges  $(e_1, \dots, e_i)$  depend only on the colorings  $(\chi_1, \dots, \chi_{i-1})$  and the randomness of the function  $f$ ; thus the transcript  $\tau$  so far depends on the color lists in  $\mathcal{Q}_<$ , but is independent of the color lists in  $\mathcal{Q}_= \cup \mathcal{Q}_>$ . It follows that conditioned on the transcript  $\tau$ , the value  $\chi_i$  of the next coloring returned is independent of whether or not a given subset  $\gamma$  of some future list  $\sigma$  of edges inserted is stored in the set  $A$ .  $\square$

**Lemma 3.2.10.** *Let  $m$  be an integer, and let  $\eta$  be an adversary for Algorithm 3 for which the first  $m$  edges submitted are always valid inputs for Algorithm 3. (In other words, no edge is repeated, and no vertex attains degree  $> L$ .) Let  $E$  be an event which depends only on the list of edges provided by  $\eta$  and the subset of those edges which Algorithm 3 stores in the set  $A$ . Then there is a specific fixed input stream of length  $m$  on which  $\Pr[E]$  is at least as large as when  $\eta$  chooses the inputs.<sup>5</sup>*

<sup>5</sup>In fact, one can prove that there is a distribution over fixed input streams so that the probability of  $E$  occurring is exactly the same as when  $\eta$  is used to pick the input.

*Proof.* Let NEXT be the function used by  $\eta$  to pick the next input based on the list of colorings produced so far, as per Section 3.2.2. We say that a partial sequence of colorings  $\rho = (\chi_1, \dots, \chi_i)$  is *pivotal* for NEXT if there exist two suffixes of  $\rho$  given by  $\pi = (\chi_1, \dots, \chi_i, \chi_{i+1}, \chi_{i+2}, \dots, \chi_j)$  and  $\pi' = (\chi_1, \dots, \chi_i, \chi'_{i+1}, \chi'_{i+2}, \dots, \chi'_j)$ , which first differ at coordinate  $i + 1$ , and where  $\text{NEXT}(\pi) \neq \text{NEXT}(\pi')$ .

If no sequence of colorings is pivotal for NEXT, then the adversary only ever submits one stream of  $m$  edges, and we are done. Otherwise, let  $\rho$  be a maximal pivotal coloring sequence for NEXT, so that there does not exist a coloring sequence  $\pi$  which has  $\rho$  as a prefix and which is also pivotal for NEXT. We will construct a modified adversary  $\tilde{\eta}$  given by  $\widetilde{\text{NEXT}}$  which behaves the same on all coloring sequences that are not extensions of  $\rho$ , which has at least the same probability of the event  $E$ , and where neither  $\rho$  nor any of its extensions is pivotal for  $\widetilde{\text{NEXT}}$ . If  $\widetilde{\text{NEXT}}$  has no pivotal sequence of colorings, we are done; if not, we can repeat this process of finding modified adversaries with fewer pivotal sequences until that is the case.

Let  $X = (X_1, \dots, X_m)$  be the random variable whose  $i$ th coordinate corresponds to the  $i$ th coloring returned by the algorithm, when the adversary is given by NEXT. Write  $X_{1..i} = (X_1, \dots, X_i)$ . Let  $\rho = (\chi_1, \dots, \chi_i)$ . Because  $\rho$  is a maximal pivotal coloring sequence for NEXT, the next coloring returned— $X_{i+1}$ —will determine the remaining  $(m - i - 1)$  edges sent by the adversary. Let  $F$  be the random variable whose value is this list of edges. For each possible value  $\sigma$  of the conditional random variable  $(X_{i+1} | X_{1..i} = \rho)$ , let  $F_\sigma$  be the list of edges sent when  $(X_{1..i}, X_{i+1}) = (\rho, \sigma)$ . By Lemma 3.2.9, conditioned on the event  $X_{1..i} = \rho$ , and on the edges  $F_\sigma$  being sent next,  $X_{i+1}$  and the event  $E$  are independent. Thus

$$\begin{aligned} \Pr[E \mid X_{1..i} = \rho] &= \mathbb{E}_{\sigma \sim X_{i+1} | X_{1..i} = \rho} \Pr[E \mid X_{1..i} = \rho, X_{i+1} = \sigma, F = F_\sigma] \\ &= \mathbb{E}_{\sigma \sim X_{i+1} | X_{1..i} = \rho} \Pr[E \mid X_{1..i} = \rho, F = F_\sigma]. \end{aligned}$$

Consequently, there is a value  $\tilde{\sigma}$  where  $\Pr[E \mid X_{1..i} = \rho, F = F_{\tilde{\sigma}}] \geq \Pr[E \mid X_{1..i} = \rho]$ . We define  $\widetilde{\text{NEXT}}$  so as to agree with  $\text{NEXT}$ , except that after the coloring sequence  $\rho$  has been received, the adversary now picks edges according to the sequence  $F_{\tilde{\sigma}}$  instead of making a choice based on  $X_{i+1}$ . This change does not reduce the probability of  $E$  (and may even increase it conditioned on  $X_{1..i} = \rho$ ). Finally, note that neither  $\rho$  nor any extension thereof is pivotal for the function  $\widetilde{\text{NEXT}}$  used by adversary  $\tilde{\eta}$ .  $\square$

**Lemma 3.2.11** (Bounding the space usage). *In the oblivious adversary setting, if a fixed stream of a graph  $G$  with maximum degree  $\Delta$  is provided to Algorithm 3, the total number of edges stored by Algorithm 3 is  $O(n(\log n)^3)$ , with high probability.*

*Proof.* We prove this by showing the maximum number of edges adjacent to any given vertex  $v$  is  $O((\log n)^3)$  with high probability. Let  $d = \deg_G(v)$ , and  $w_1, \dots, w_d$  be the neighbors of  $v$  in  $G$ , ordered by the order in which the edges  $\{v, w_i\}$  occur in the stream. For any  $x \in [n]$ , write  $P_x$  to be the random variable consisting of all of  $x$ 's color lists,  $P_x := (P_x^i)_{i \in [L]}$ . Then for  $i \in [d]$ , define the indicator random variable  $Y_i$  to be 1 iff the algorithm records edge  $\{v, w_i\}$ ; since  $Y_i$  is determined by  $P_v$  and  $P_{w_i}$ , the random variables  $(Y_i)_{i \in [d]}$  are conditionally independent given  $P_v$ .

Now, for each  $i \in [d]$ ,

$$\begin{aligned}
 \Pr[Y_i = 1 \mid P_v] &= \Pr \left[ \bigvee_{j=1}^L \{P_{w_i}^j \cap P_v^j \neq \emptyset\} \mid P_v \right] \\
 &\leq \sum_{j=1}^L \Pr [P_{w_i}^j \cap P_v^j \neq \emptyset \mid P_v] = \sum_{j=1}^L \Pr [\exists h \in [4 \log n] : P_{w_i}^j[h] \in P_v^j \mid P_v] \\
 &\leq \sum_{j=1}^L \sum_{h=1}^{4 \log n} \Pr [P_{w_i}^j[h] \in P_v^j \mid P_v] = \sum_{j=1}^L 4 \log n \cdot \frac{|P_v^j|}{2j^2} \\
 &\leq 16(\log n)^2 \sum_{j=1}^{\infty} \frac{1}{j(j+1)} = \frac{16(\log n)^2}{i}.
 \end{aligned}$$

Since  $\mathbb{E}[Y_i \mid P_v] = \Pr[Y_i = 1 \mid P_v]$ , this upper bound gives

$$\mathbb{E} \left[ \sum_{i=1}^d Y_i \mid P_v \right] \leq \sum_{i=1}^d \frac{16(\log n)^2}{i} \leq 32(\log n)^3,$$

using the fact that  $\sum_{i=1}^d 1/i \leq \max\{2 \log d, 1\} \leq 2 \log n$ . Applying a form of the Chernoff bound:

$$\Pr \left[ \sum_{i=1}^d Y_i \geq 2 \cdot 32(\log n)^3 \mid P_v \right] \leq \exp \left( -\frac{1}{3} \cdot 32(\log n)^3 \right) \leq \frac{1}{n^3},$$

which proves that the number of edges adjacent to  $v$  is  $\leq 64(\log n)^3$  with high probability, for any value of  $P_v$ .

Applying a union bound over all  $n$  vertices, the probability that the maximum degree of the stored graph  $A$  exceeds  $64(\log n)^3$  is less than  $1/n^2$ .  $\square$

Combining Lemma 3.2.8, Lemma 3.2.10 and Lemma 3.2.11, we arrive at the main result of this section.

**Theorem 3.2.12.** *Algorithm 3 is an adversarially robust  $O(\Delta^3)$ -coloring algorithm for insertion streams which stores  $O(n(\log n)^4)$  bits related to the graph, requires access to  $\tilde{O}(nL)$  random bits, and even against an adaptive adversary succeeds with probability  $\geq 1 - O(1/n^2)$ .*  $\square$

A weakness of Algorithm 3 is that it requires the algorithm be able to access all  $\tilde{O}(nL)$  random bits in advance. If we assume that the adversary is limited in some fashion, then it may be possible to store  $\leq \tilde{O}(n)$  true random bits, and use a pseudorandom number generator to produce the  $\tilde{O}(nL)$  bits that the algorithm uses, on demand. For example, if the adversary only can use  $O(n/\log n)$  bits of space, using Nisan's PRG [146] on  $\Omega(n)$  true random bits will fool the adversary. Alternatively, assuming one-way functions exist, there is a classic construction [101] to produce a pseudorandom number generator using

$O(n)$  true random bits, which in polynomial time generates  $\text{poly}(n)$  pseudorandom bits that any adversary limited to using polynomial time cannot distinguish with non-negligible probability from truly random bits.

### 3.2.7. Sketch-Switching Based Algorithms for Turnstile Streams

We present a class of sketch switching based algorithms for  $\text{poly}(\Delta)$ -coloring. First, we give an outline of a simple algorithm for insert-only streams that colors the graph using  $O(\Delta^2)$  colors and  $\tilde{O}(n\sqrt{\Delta})$  space, where  $\Delta$  is the max-degree of the graph at the time of query. Next, we show how to modify it to handle deletions. This is given by Algorithm 4, whose correctness is proven in Lemma 3.2.15. Then we describe how it can be generalized to get an  $O(\Delta^k)$ -coloring in  $\tilde{O}(n\Delta^{1/k})$  space for insert-only streams for any constant  $k \in \mathbb{N}$ . Finally, we prove the fully general result giving an  $O(\Delta^k)$ -coloring in  $\tilde{O}(n^{1-1/k}m^{1/k})$  space for turnstile streams, which is given by Theorem 3.2.16. Finally, we discuss how we can get rid of some reasonable assumptions that we make for our algorithms and how to improve the query time.

Throughout this section, we make the standard assumption that the stream length  $m$  for turnstile streams is bounded by  $\text{poly}(n)$ . When we say that a statement holds with high probability (w.h.p.), we mean that it holds with probability at least  $1 - 1/\text{poly}(n)$ . In our algorithms, we often take the *product* of colorings of multiple subgraphs of a graph  $G$ . We define this notion below and record its key property.

**Definition 3.2.13** (Product of Colorings). Let  $G_1 = (V, E_1), \dots, G_k = (V, E_k)$  be graphs on a common vertex set  $V$ . Given a coloring  $\chi_i$  of  $G_i$ , for each  $i \in [k]$ , the *product* of these colorings is defined to be a coloring where each vertex  $v \in V$  is assigned the color  $\langle \chi_1(v), \chi_2(v), \dots, \chi_k(v) \rangle$ .

**Lemma 3.2.14.** *Given a proper  $c_i$ -coloring  $\chi_i$  of a graph  $G_i = (V, E_i)$  for each  $i \in [k]$ , the product of the colorings  $\chi_i$  is a proper  $(\prod_{i=1}^k c_i)$ -coloring of  $\cup_{i=1}^k G_i := (V, \cup_{i=1}^k E_i)$ .*

*Proof.* An edge in  $\cup_{i=1}^k G_i$  comes from  $G_{i^*}$  for some  $i^* \in [k]$ , and hence the colors of its endpoints in the product coloring differ in the  $i^*$ -th coordinate. For  $i \in [k]$ , the  $i$ -th coordinate can take  $c_i$  different values and hence the color bound holds.  $\square$

**Insert-Only Streams and  $O(\Delta^2)$ -Coloring.** Split the  $O(n\Delta)$ -length stream into  $\sqrt{\Delta}$  chunks of size  $O(n\sqrt{\Delta})$  each. Let  $\mathcal{A}$  be a standard (i.e., oblivious-adversary) semi-streaming algorithm for  $O(\Delta)$ -coloring a graph (by Fact 3.2.1 and Fact 3.2.2, such algorithms exist). At the start of the stream, initialize  $\sqrt{\Delta}$  parallel copies of  $\mathcal{A}$ , called  $A_1, \dots, A_{\sqrt{\Delta}}$ ; these will be our “parallel sketches.” At any point of time, only a suffix of this list of parallel sketches will be active.

We use the sketch switching idea of [40] as follows. With each edge insertion, we update each of the active parallel sketches. Whenever we arrive at the end of a chunk, we say we have reached a “checkpoint” and query the least-numbered active sketch (this is guaranteed to be “fresh” in the sense that it has not been queried before) to produce a coloring of the entire graph until that point. By design, the randomness of the queried sketch is independent of the edges it has processed. Therefore, it returns a correct  $O(\Delta)$ -coloring of the graph until that point, w.h.p. Henceforth, we mark the just-queried sketch as inactive and never update it, but continue to update all higher-numbered sketches. Thus, each copy of  $\mathcal{A}$  actually processes a stream independent of its randomness and hence, works correctly while using  $\tilde{O}(n)$  space. By a union bound over all sketches, w.h.p., all of them generate correct colorings at the respective checkpoints and simultaneously use  $\tilde{O}(n)$  space each, i.e.,  $\tilde{O}(n\sqrt{\Delta})$  space in total.

Conditioned on the above good event, we can always return an  $O(\Delta^2)$ -coloring as follows. We store (buffer) the most recent partial chunk explicitly, using our available  $\tilde{O}(n\sqrt{\Delta})$  space. Now, when a query arrives, we can express the current graph  $G$  as  $G_1 \cup G'$ , where  $G_1$  is the subgraph of  $G$  until the last checkpoint and  $G'$  is the subgraph in our buffer. Observe that we computed an  $O(\Delta(G_1))$ -coloring of  $G_1$  at the last checkpoint. Further, we

can deterministically compute a  $(\Delta(G') + 1)$ -coloring of  $G'$  since we explicitly store it. We output the product of the colorings (Definition 3.2.13) of  $G_1$  and  $G'$ , which must be a proper  $O(\Delta(G_1) \cdot \Delta(G')) = O(\Delta(G)^2)$ -coloring of the graph  $G$  (Lemma 3.2.14).

**Extension to Handle Deletions.** The algorithm above doesn't immediately work for turnstile streams. The chunk currently being processed by the algorithm may contain an update that deletes an edge which was inserted before the start of the chunk, and hence, is not in store. Thus, we can no longer express the current graph as a union of the graphs  $G_1$  and  $G'$  as above. Overcoming this difficulty complicates the algorithm enough that it is useful to lay it out more formally as pseudocode (see Algorithm 4). This new algorithm maintains an  $O(\Delta^2)$ -coloring, works even on turnstile streams, and uses  $\tilde{O}(\sqrt{mn})$  space. Note that while the blackbox algorithm  $\mathcal{A}$  used in Algorithm 4 might be any generic  $O(\Delta)$ -coloring semi-streaming algorithm with error  $1/m$ , it can be, for instance, chosen to be the one given by Fact 3.2.1 or, for insert-only streams, the one in Fact 3.2.2. The former gives a tight  $(\Delta + 1)$ -coloring but possibly large query time, while the latter answers queries fast using possibly a few more colors, up to  $\Delta(1 + \varepsilon)$ .<sup>6</sup>

Before proceeding to the analysis, let us set up some terminology. Recall from Section 3.2.2 that we work with strict graph turnstile streams, so each deletion of an edge  $e$  can be matched to a unique previous token that most recently inserted  $e$ . An edge deletion, where the corresponding insertion did not occur inside the same chunk, is called a *negative edge*. Call a point in the stream a *checkpoint* if we use a *fresh* parallel copy of  $\mathcal{A}$ , i.e., a copy  $A_i$  that hasn't been queried before, to generate an  $O(\Delta)$ -coloring of the graph at that point. We define two types of checkpoints, namely *fixed* and *ad hoc*. We have a fixed checkpoint at the end of each chunk; this means that whenever the last update of a chunk arrives, we compute a coloring of the graph seen so far using a fresh copy of  $\mathcal{A}$ . The ad hoc checkpoints are made on the fly inside a current chunk, precisely when a query appears and

<sup>6</sup>In practice, however, the latter uses significantly fewer colors for most graphs since it's a  $\kappa(1 + \varepsilon)$ -coloring algorithm and  $\kappa \leq \Delta$  always, and, in fact,  $\kappa \ll \Delta$  for real world graphs. [43]

---

**Algorithm 4** Adversarially robust  $O(\Delta^2)$ -coloring in  $\tilde{O}(\sqrt{nm})$  space for turnstile streams

---

**Input:** Stream of edge insertions/deletions of  $n$ -vertex graph  $G = (V, E)$ ; parameter  $m$

**Require:** Semi-streaming algorithm  $\mathcal{A}$  that works on turnstile graph streams and provides an  $O(\Delta)$ -coloring with error  $\leq 1/m$  against an oblivious adversary

**Initialize:**

- 1:  $s \leftarrow C \cdot \sqrt{m/n} \log n$  for some sufficiently large constant  $C$
- 2:  $A_1, \dots, A_s \leftarrow$  independent parallel initializations of  $\mathcal{A}$
- 3:  $c \leftarrow 0$  ▷ index into list  $(A_1, \dots, A_s)$
- 4:  $\text{CLR} \leftarrow n$ -vector of vertex colors, initialized to all-1s ▷ valid  $O(\Delta)$ -coloring until last checkpoint
- 5:  $\text{DEG} \leftarrow n$ -vector of vertex degrees, initialized to all-0s
- 6:  $G' \leftarrow (V, \emptyset)$  ▷ buffer to store current chunk
- 7:  $\text{CHUNKSIZE} \leftarrow 0$  ▷ current buffer size
- 8:  $\text{CHECKPTMAXDEG} \leftarrow 0$  ▷ max-degree at last checkpoint

**Process(operation OP, edge  $\{u, v\}$ ):**

▷ OP says whether to insert or delete

- 9: **for**  $i$  from  $c + 1$  to  $s$  **do**
- 10:      $A_i.$ **Process**(OP,  $\{u, v\}$ ) ▷ if this aborts, report FAIL
- 11: **if** OP = “insert” **then**
- 12:     increment  $\text{DEG}(u), \text{DEG}(v)$
- 13:     add  $\{u, v\}$  to  $G'$
- 14: **else if** OP = “delete” **then**
- 15:     decrement  $\text{DEG}(u), \text{DEG}(v)$
- 16:     **if**  $\{u, v\} \in G'$  **then:** ▷ else, negative edge; not stored
- 17:         delete  $\{u, v\}$  from  $G'$
- 18:  $\text{CHUNKSIZE} \leftarrow \text{CHUNKSIZE} + 1$
- 19:  $\Delta \leftarrow \max_{v \in [n]} \text{DEG}(v)$
- 20: **if**  $\text{CHUNKSIZE} = \sqrt{nm}$  **then:**
- 21:     **NewCheckpoint**( ) ▷ fixed checkpoint encountered
- 22:      $\text{CHUNKSIZE} \leftarrow 0$
- 23: **if**  $\Delta < \text{CHECKPTMAXDEG}/2$  **then:**
- 24:     **NewCheckpoint**( ) ▷ ad hoc checkpoint created

**Query( ):**

- 25:  $\text{CLR}' \leftarrow (\Delta_{G'} + 1)$ -coloring of  $G'$
- 26: **return**  $\langle (\text{CLR}(v), \text{CLR}'(v)) : v \in [n] \rangle$  ▷ take the product of the two colorings

**NewCheckpoint( ):**

- 27:  $c \leftarrow c + 1$  ▷ switch to next fresh sketch
  - 28:  $\text{CLR} \leftarrow A_c.$ **Query**( ) ▷ if  $A_c$  fails, report FAIL
  - 29:  $G' \leftarrow (V, \emptyset)$
  - 30:  $\text{CHECKPTMAXDEG} \leftarrow \max_{v \in [n]} \text{DEG}(v)$
-

we see that the max-degree of the current graph is less than half of what it was at the *last* checkpoint (which might be fixed or ad hoc). We now analyze Algorithm 4 in the following lemma.

**Lemma 3.2.15.** *For any strict graph turnstile stream of length at most  $m$  for a graph  $G$  given by an adaptive adversary, the following hold simultaneously, w.h.p.:*

- (i) *Algorithm 4 outputs an  $O(\Delta^2)$ -coloring after each query, where  $\Delta$  is the maximum degree of the graph at the time a query is made.*
- (ii) *Algorithm 4 uses  $\tilde{O}(\sqrt{mn})$  bits of space.*

*Proof.* Notice that Algorithm 4 splits the stream into chunks of size  $\sqrt{mn}$ . It processes one chunk at a time by explicitly storing all updates in it except for the negative edges. Nevertheless, when a negative edge arrives, the chunk size increases and importantly, we do update the appropriate copies of  $\mathcal{A}$  with it. Buffer  $G'$  maintains the graph induced by the updates stored from the current chunk. The counter  $c$  maintains the number of (overall) checkpoints reached. Whenever we reach a checkpoint, we re-initialize  $G'$  to  $G_0$ , defined as the empty graph on the vertex set  $V$ . For  $c \geq 1$ , let  $G_c$  denote the graph induced by all updates until checkpoint  $c$ .

Note that answers to all queries (if any) that are made following some update before checkpoint  $c$  depends only on sketches  $A_i$  for some  $i < c$  (if any). Thus, the random string used by the sketch  $A_c$  is independent of the graph  $G_c$ . Hence, by the correctness guarantees of algorithm  $\mathcal{A}$ , the copy  $A_c$  produces a valid  $O(\Delta)$ -coloring CLR of  $G_c$  with probability at least  $1 - 1/m$ . Furthermore, observe that an edge update before checkpoint  $c$  is dependent on only the outputs of the sketches  $A_j$  for  $j < c$ . However, we insert such an update only to copies  $A_i$  for  $i \geq c$ . Therefore, the random string of any sketch  $A_i$  is independent of the graph edges it processes. Thus, by the space guarantees of algorithm  $\mathcal{A}$ , a sketch  $A_i$  uses  $\tilde{O}(n)$  space with probability  $1 - 1/m$ . By a union bound over all  $s = O(\sqrt{m/n} \log n)$

copies, with probability at least  $1 - 1/\text{poly}(n)$ , for all  $c \in [s]$ , the sketch  $A_c$  produces a valid  $O(\Delta)$ -coloring of the graph  $G_c$  and uses  $\tilde{O}(n)$  space. Now, conditioning on this event, we prove that (i) and (ii) always hold. Hence, in general, they hold with probability at least  $1 - 1/\text{poly}(n)$ .

Consider a query made at some point in the stream. Since we keep track of all the vertex degrees and save the max-degree at the last checkpoint, we can compare the max-degree  $\Delta$  of the current graph  $G$  with  $\Delta(G_c)$ , where  $c$  is the last checkpoint (can be fixed or ad hoc). In case  $\Delta < \Delta(G_c)/2$ , we declare the current query point as an ad hoc checkpoint  $c + 1$ , i.e., we use the next fresh sketch  $A_{c+1}$  to compute an  $O(\Delta)$ -coloring of the current graph  $G_{c+1}$ . Since we encounter a checkpoint, we reset CLR to this coloring and  $G'$  to  $G_0$ , implying that CLR' is just a 1-coloring of the empty graph. Thus, the product of CLR and CLR' that is returned uses only  $O(\Delta)$  colors and is a proper coloring of the graph  $G_{c+1}$ .

In the other case that  $\Delta \geq \Delta(G_c)/2$ , we output the coloring obtained by taking a product of the  $O(\Delta(G_c))$ -coloring CLR at the last checkpoint  $c$  and a  $(\Delta(G') + 1)$ -coloring CLR' of the graph  $G'$ . Note that we can obtain the latter deterministically since we store  $G'$  explicitly. Observe that the edge set of the graph  $G$  is precisely  $(E(G_c) \setminus F) \cup E(G')$ , where  $F$  is the set of negative edges in the current chunk. Since the coloring we output is a proper coloring of  $G_c \cup G'$  (Lemma 3.2.14), it must be a proper coloring of  $G$  as well because edge deletions can't violate it. It remains to prove the color bound. The number of colors we use is at most  $O(\Delta(G_c) \cdot \Delta(G'))$ . We have checked that  $\Delta \geq \Delta(G_c)/2$ . Again, observe that  $\Delta(G') \leq \Delta$  since  $G'$  is a subgraph of  $G$ . Therefore, the number of colors used is at most  $O(2\Delta \cdot \Delta) = O(\Delta^2)$ .

To complete the proof that (i) holds, we need to ensure that before the stream ends, we don't introduce too many ad hoc checkpoints so as to run out of fresh sketches to invoke at the checkpoints. We declare a point as an ad hoc checkpoint only if the max-degree has fallen below half of what it was at the last checkpoint (fixed or ad hoc). Therefore,

along the sequence of ad hoc checkpoints between two consecutive fixed checkpoints (i.e., inside a chunk), the max-degree decreases by a factor of at least 2. Hence, there can be only  $O(\log \Delta_{\max}) = O(\log n)$  ad hoc checkpoints inside a single chunk, where  $\Delta_{\max}$  is the maximum degree of a vertex over all intermediate graphs in the stream. We have  $O(\sqrt{m/n})$  chunks and hence,  $O(\sqrt{m/n})$  fixed checkpoints and at most  $O(\sqrt{m/n} \log n)$  ad hoc checkpoints. Thus, the total number of checkpoints is at most  $s = O(\sqrt{m/n} \log n)$  and it suffices to have that many sketches initialized at the start of the stream.

To verify (ii), note that since each chunk has size  $\sqrt{mn}$ , we use at most  $\tilde{O}(\sqrt{mn})$  bits of space to store  $G'$ . Also, each of the  $s$  parallel sketches takes  $\tilde{O}(n)$  space, implying that they collectively use  $\tilde{O}(ns) = \tilde{O}(\sqrt{mn})$  space. Storing all the vertex degrees takes  $\tilde{O}(n)$  space. Therefore, the total space usage is  $\tilde{O}(\sqrt{mn})$  bits.  $\square$

**Generalization to  $O(\Delta^k)$ -Coloring in  $\tilde{O}(n\Delta^{1/k})$  Space for Insert-Only Streams.** We aim to generalize the above result by attaining a color-space tradeoff. Again, for insert-only streams, it is not hard to obtain such a generalization and we outline the algorithm for this setting first. Algorithm 4 shows that we need to use roughly  $\tilde{O}(nr)$  space if we split the stream into  $r$  chunks since we use a fresh  $\tilde{O}(n)$ -space sketch at the end of each chunk. Thus, to reduce the space usage, we can split the stream into smaller number of chunks. However, that would make the size of each chunk larger than our target space bound. Hence, instead of storing it entirely and coloring it deterministically as before, we treat it as a smaller stream in itself and recursively color it using space smaller than its size. To be precise, suppose that for any  $d$ , we can color a stream of length  $nd$  using  $O(\Delta^\ell)$  colors and  $\tilde{O}(nd^{1/\ell})$  space for some integer  $\ell$  (this holds for  $\ell = 2$  by Lemma 3.2.15). Now, suppose we split an  $nd$ -length stream into  $d^{1/(\ell+1)}$  chunks of size  $nd^{\ell/(\ell+1)}$ . We use a fresh sketch at each chunk end or checkpoint to compute an  $O(\Delta)$ -coloring of the graph seen so far. We can then recursively color the subgraph induced by each chunk using  $O(\Delta^\ell)$  colors and  $\tilde{O}\left(n\left(d^{\ell/(\ell+1)}\right)^{1/\ell}\right) = \tilde{O}(nd^{1/(\ell+1)})$  space. As before, taking a product of this coloring

with an  $O(\Delta)$ -coloring at the last checkpoint gives an  $O(\Delta^{\ell+1})$ -coloring (Lemma 3.2.14) of the current graph in  $\tilde{O}(nd^{1/(\ell+1)})$  space. The additional space used by the parallel sketches for the  $d^{1/(\ell+1)}$  many chunks is also  $\tilde{O}(nd^{1/(\ell+1)})$ . Therefore, by induction, we can get an  $O(\Delta^k)$ -coloring in  $\tilde{O}(nd^{1/k}) = O(n\Delta^{1/k})$  space for any integer  $k$ . We capture this result in Corollary 3.2.17 after proving the more general result for turnstile streams.

**Fully General Algorithm for Turnstile Streams.** Handling edge deletions with the above algorithm is challenging because of the same reason as earlier: a chunk of the stream may not itself represent a subgraph as it can have negative edges. Therefore, it is not clear that we can recurse on that chunk with a blackbox algorithm for a graph stream. A trick to handle deletions as in Algorithm 4 faces challenges due to the recursion depth. We shall have an  $O(\Delta)$ -coloring at a checkpoint at each level of recursion that we basically combine to obtain the final coloring. Previously, we checked whether the max-degree has decreased significantly since the last checkpoint and if so, declared it as an ad hoc checkpoint. This time, due to the presence of checkpoints at multiple recursion levels, if the  $\Delta$ -value is too high at even a single level, we need to have an ad hoc checkpoint, which might turn out to be too many. We show how to extend the earlier technique to overcome this challenge and obtain the general result for turnstile streams, which achieves an  $O(\Delta^k)$ -coloring in  $\tilde{O}(n^{1-1/k}m^{1/k})$  space for an  $m$ -length stream.

**Theorem 3.2.16.** *For any strict graph turnstile stream of length at most  $m$ , and for any constant  $k \in \mathbb{N}$ , there exists an adversarially robust algorithm  $\mathcal{A}$  such that the following hold simultaneously w.h.p.:*

- (i) *After each query,  $\mathcal{A}$  outputs an  $O(\Delta^k)$ -coloring, where  $\Delta$  is the max-degree of the current graph.*
- (ii)  *$\mathcal{A}$  uses  $\tilde{O}(n^{1-1/k}m^{1/k})$  bits of space.*

*Proof.* The following framework is an extension of Algorithm 4 that would be given by the

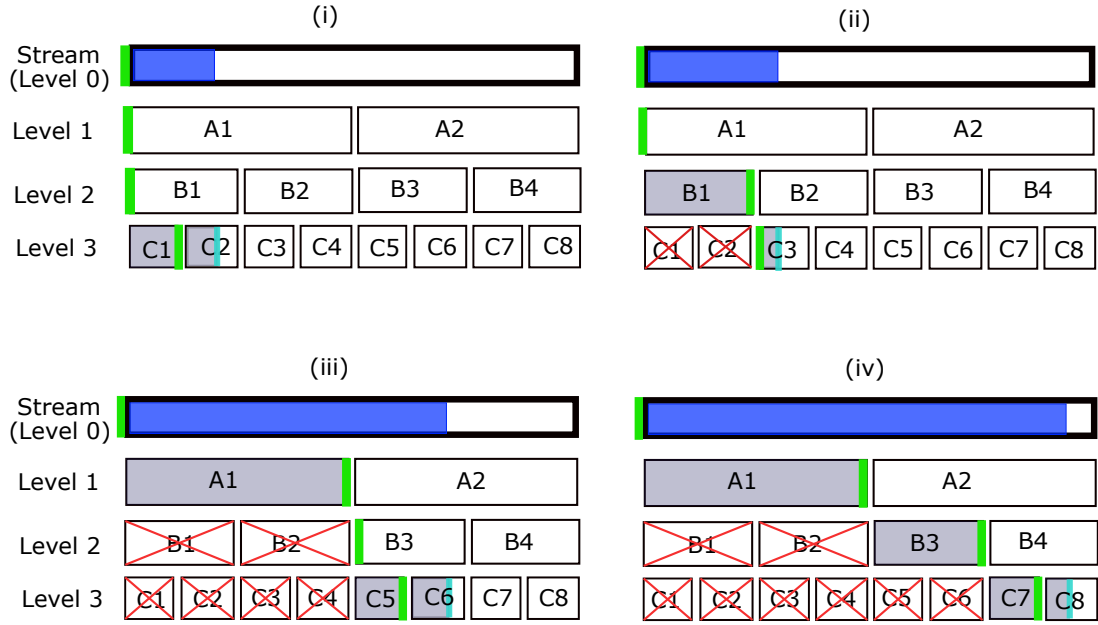


Figure 3.1: Certain states of the data structure of our  $O(\Delta^k)$ -coloring algorithm for  $k = 4$ . We pretend that we always split into  $d^{1/k} = 2$  chunks. The stream is a level-0 chunk;  $A_1, A_2$  are level-1 chunks;  $B_1, \dots, B_4$  are level-2; and  $C_1, \dots, C_8$  are level-3. For each state, the top blue bar shows the progress of the stream. Each level has a green vertical bar that represents the last checkpoint in that level. The chunks filled in gray represent the subgraphs defined as  $G_i$ . A partially filled chunk (endpoint colored cyan) is the current chunk from which the subgraph  $G'$  is stored. A chunk is crossed out in red if it has been subsumed by a higher level chunk.

recursion idea discussed above. Figure 3.1 shows the setup of our data structure. The full stream is the sole “level-0” chunk. Given  $k$ , we first split the edge stream into  $d^{1/k}$  chunks of size  $O(nd^{(k-1)/k})$  each, where  $d = m/n$ : these chunks are in “level 1.” For  $1 \leq i \leq k - 2$ , recursively split each level- $i$  chunk into  $d^{1/k}$  subchunks of size  $O(nd^{(k-i-1)/k})$  each, which we say are in level  $i + 1$ . Level  $k - 1$  thus has chunks of size  $O(nd^{1/k})$ . We explicitly store all updates in a level- $(k - 1)$  chunk except the negative edges, one chunk at a time.

Let  $A$  be a turnstile streaming algorithm in the oblivious adversary setting that uses at most  $\Delta(1 + \varepsilon)$  colors, where  $\varepsilon = 1/2k$ , and  $\tilde{O}(n)$  space, and fails with probability at most  $1/(mn)$ . By Fact 3.2.1, such an algorithm exists.<sup>7</sup> At the start of the stream, for

<sup>7</sup>By Fact 3.2.2, another algorithm with these properties exists for insert-only streams.

each  $i \in [k - 1]$ , we initialize  $s = O(d^{1/k}(k \log n)^k)$  parallel copies or “level- $i$  sketches”  $A_{i,1}, \dots, A_{i,s}$  of  $A$ . For each  $i$ , the level- $i$  sketches process the level- $i$  chunks. Henceforth, over the course of the stream, as soon as we reach the end of a level- $i$  chunk, since it subsumes all its subchunks, we re-initialize the level- $j$  sketches for each  $j > i$ . As before, at the end of each chunk in each level  $i$ , we have a “checkpoint”, i.e., we query a fresh level- $i$  sketch  $A_{i,r}$  for some  $r \in [s]$  to compute a coloring at such a point. Observe that this is a coloring of the subgraph starting from the last level- $(i - 1)$  checkpoint through this point. Following previous terminology, we call these level- $i$  chunk ends as *fixed* “level- $i$  checkpoints”. (For instance, in Figure 3.1, in (i), the checkpoint at the end of chunk  $C1$  is a fixed level-3 checkpoint, while in (iii), the checkpoint at the end of  $A1$  is a fixed level-1 checkpoint.)

This time, we can also have what we call *vacuous* checkpoints. The start of the stream is a vacuous level- $i$  checkpoint for each  $0 \leq i \leq k - 1$ . Further, for each  $i \in [k - 2]$ , after the end of each level- $i$  chunk, i.e., immediately after a fixed level- $i$  checkpoint, we create a vacuous level- $j$  checkpoint for each  $j > i$  (e.g., in Figure 3.1, in (i), the checkpoint at the start of  $B1$  is a vacuous level-2 checkpoint, while in (ii), the one at the start of  $C3$  is a vacuous level-3 checkpoint). It is, after all, a level- $j$  “checkpoint”, so we want a coloring stored for the substream starting from the last level- $(j - 1)$  checkpoint through this point. However, note, that for each  $j > i$  this substream is empty (hence the term “vacuous”). Hence, we don’t waste a sketch for a vacuous checkpoint and directly store a 1-coloring for that empty substream.

We can also have *ad hoc* level- $i$  checkpoints that we declare on the fly (when to be specified later). Just as we would do on reaching a fixed level- $i$  checkpoint, we do the following upon creating an ad hoc level- $i$  checkpoint: (i) query a fresh level- $i$  sketch to compute a coloring at this point (again, this is a coloring of the subgraph from the last level- $(i - 1)$  checkpoint until this point), (ii) start splitting the remainder of the stream into

subchunks of higher levels, (iii) re-initialize the level- $j$ -sketches for each  $j > i$ , and (iv) create vacuous level- $j$  checkpoints for each  $j > i$ .

Any copy of algorithm  $A$  that we use in any level is updated and queried as in Algorithm 4: we update each copy as long as it is not used to answer a query of the adversary and whenever we query a sketch, we make sure that it has not been queried before. Therefore, as in Algorithm 4, the random string of any copy is independent of the graph edges it processes. Hence, each sketch computes a coloring correctly and uses  $\tilde{O}(n)$  space with probability at least  $1 - 1/(mn)$ . Taking a union bound over all  $O(ds) = \tilde{O}(d^{1+1/k})$  sketches, we get that all of them simultaneously provide correct colorings and use  $\tilde{O}(n)$  space each with probability at least  $1 - 1/\text{poly}(n)$ . Henceforth, as in the proof of Lemma 3.2.15, we condition on this event and show that (i) and (ii) always hold, thus proving that they hold w.h.p. in general.

For  $1 \leq i \leq k - 1$ , define  $G_i$  as the graph starting from the last level- $(i - 1)$  checkpoint through the last level- $i$  checkpoint (in Figure 3.1, the last checkpoint in each level is denoted by a green bar, and the  $G_i$ 's are the graphs between two such consecutive bars; they are either filled with gray or empty; for instance, in (ii),  $G_1 = \emptyset$ ,  $G_2 = B1$ , and  $G_3 = \emptyset$ , while in (iv),  $G_1 = A1$ ,  $G_2 = B3$ , and  $G_3 = C7$ ). Note that a graph  $G_i$  might be empty: this happens when the last level- $i$  checkpoint is vacuous. Observe that we can express the current graph  $G$  as  $((G_1 \cup G_2 \cup \dots \cup G_{k-1}) \setminus F) \cup G'$ , where,  $G'$  is the subgraph stored from the the current chunk in level  $(k - 1)$  (recall that it is induced by all updates in this chunk excluding the negative edges), and  $F$  is the set of negative edges in the chunk. It is easy to see that we can keep track of the degrees so that we know  $\Delta(G_i)$  for each  $i$ . We check whether there exists an  $i \in [k - 1]$  such that the max-degree  $\Delta$  of the current graph  $G$  is less than  $\Delta(G_i)/(1 + \varepsilon)$ . If not, we take the coloring from the last checkpoint of each level in  $[k - 1]$  and return the product of all these colorings with a  $(\Delta(G') + 1)$ -coloring of  $G'$  (Definition 3.2.13). We can compute the latter deterministically since we have  $G'$  in

store. Notice that the colorings at the checkpoints are valid colorings of  $G_i$  for  $i \in [k-1]$  using 1 color if  $G_i$  is empty and at most  $(1+\varepsilon)\Delta(G_i) \leq (1+\varepsilon)^2\Delta$  colors otherwise. Also,  $\Delta(G') \leq \Delta$  because  $G'$  is a subgraph of  $G$ . Therefore, by Lemma 3.2.14, the total number of colors used to color  $G$  is

$$\prod_{i=1}^{k-1} (\max\{(1+\varepsilon)^2\Delta, 1\}) \cdot (\Delta+1) \leq O((1+\varepsilon)^{2k-2}\Delta^k) = O(\Delta^k),$$

since  $2k-2 < 2k = 1/\varepsilon$ . Finally, note that the product obtained will be a proper coloring of  $G$  since the negative edges in  $F$  cannot violate it.

In the other case that there exists an  $i$  such that  $\Delta < \Delta(G_i)/(1+\varepsilon)$ , let  $i^*$  be the first such  $i$ . We make this query point an ad hoc level- $i^*$  checkpoint. Also, the graph  $G_{i^*}$  changes according to the definition above, and now the current graph  $G$  is given by  $G_1 \cup \dots \cup G_{i^*}$ . Then, we return the product of colorings at the last checkpoints of levels  $1, \dots, i^*$ . We know that these give  $(1+\varepsilon)\Delta(G_i)$ -colorings for  $i \in [i^*]$ . Again, we have  $\Delta(G_i) \leq \Delta$  since  $G_i$  is a subgraph of  $G$  for each  $i$ . Thus, the total number of colors used is

$$\prod_{i=1}^{i^*} ((1+\varepsilon)\Delta(G_i)) = (1+\varepsilon)^{i^*} \Delta^{i^*} = O(\Delta^{k-1}),$$

since  $i^* \leq k-1 < 1/2\varepsilon$ . Therefore, in either case, we get an  $O(\Delta^k)$ -coloring.

Now, as in the proof of Lemma 3.2.15, we need to prove that we have enough parallel sketches for the ad hoc checkpoints. Observe that we create an ad hoc level- $i$  checkpoint only when the current max-degree decreases by a factor of  $(1+\varepsilon)$  from the last checkpoint in level  $i$  itself. Thus, along the sequence of ad hoc level- $i$  checkpoints between two consecutive non-ad-hoc (fixed or vacuous) level- $i$  checkpoints, the max-degree decreases by a factor of at least  $(1+\varepsilon)$ . Therefore, there can be at most  $\log_{1+\varepsilon} n = O(\varepsilon^{-1} \log n) = O(k \log n)$  such ad hoc checkpoints.

We show by induction that the number of ad hoc checkpoints in any level  $i$  is  $O(d^{1/k}(k \log n)^i)$ .

In level 1, there is only 1 vacuous checkpoint (at the beginning) and  $d^{1/k}$  fixed checkpoints. Therefore, by the argument above, it can have  $O(d^{1/k}(k \log n))$  ad hoc checkpoints; the base case holds. By induction hypothesis assume that it is true for all  $i \leq j$ . The number of vacuous checkpoints in level  $j$  is equal to the number of fixed plus ad hoc checkpoints in levels  $1, \dots, j-1$ . This is  $\sum_{i=1}^{j-1} O(d^{1/k}(k \log n)^i) = O(d^{1/k} k^j \log^{j-1} n)$  since  $j < k$ . The number of ad hoc checkpoints in level  $j$  is  $\log n$  times the number of vacuous plus fixed checkpoints in level  $j$ , which is  $O(d^{1/k} k^j \log^{j-1} n \cdot \log n) = O(d^{1/k}(k \log n)^j)$ . Thus, by induction, there are  $O(d^{1/k}(k \log n)^i)$  ad hoc checkpoints in any level  $i$ . Therefore, the total number of checkpoints in level  $i$  is also  $O(d^{1/k}(k \log n)^i + d^{1/k} k^i \log^{i-1} n + d^{1/k}) = O(d^{1/k}(k \log n)^i)$ . Thus,  $s = O(d^{1/k}(k \log n)^k)$  many parallel sketches suffice for each level. This completes the proof of (i).

Finally, for (ii), as noted above, the  $s$  parallel sketches of  $A$  take up  $\tilde{O}(n)$  space individually, and hence,  $\tilde{O}(ns) = \tilde{O}(nd^{1/k})$  space in total (recall that  $k = O(1)$ ). Additionally, the space usage to store the subgraph  $G'$  from a level- $(k-1)$  chunk is  $\tilde{O}(nd^{1/k})$ . Hence, the total space used is  $\tilde{O}(nd^{1/k}) = \tilde{O}(n^{1-1/k}m^{1/k})$ .  $\square$

The next corollary shows that the space bound for  $O(\Delta^k)$ -coloring on insert-only streams is  $\tilde{O}(n\Delta^{1/k})$  and follows immediately from Theorem 3.2.16 noting that  $m = O(n\Delta)$  for such streams. Note that it works even for  $k = \omega(1)$  since we don't have ad hoc checkpoints for insert-only streams and just  $d^{1/k}$  sketches per level suffice.

**Corollary 3.2.17.** *For any stream of edge insertions describing a graph  $G$ , and for any  $k \in \mathbb{N}$ , there exists an adversarially robust algorithm  $\mathcal{A}$  such that the following hold simultaneously w.h.p.:*

- *After each query,  $\mathcal{A}$  outputs an  $O(\Delta^k)$ -coloring, where  $\Delta$  is the max-degree of the current graph.*
- *$\mathcal{A}$  uses  $\tilde{O}(n\Delta^{1/k})$  bits of space.*  $\square$

**Implementation Details: Update and Query Time.** Observe that if we use the algorithm by [17] or [43] as a blackbox, then, to answer each query of the adversary, the time we spend is the post-processing time of these algorithms, which are  $\tilde{O}(n\sqrt{\Delta})$  and  $\tilde{O}(n)$  respectively. Although in the streaming setting, we don't care that much about the time complexity, such a query time might be infeasible in practice since we can potentially have a query at every point in the stream. Thus, ideally, we want an algorithm that *maintains* a coloring at every point in the stream spending a reasonably small time to update the solution after each edge insertion/deletion. This is similar to the dynamic graph algorithms setting, except here, we are asking for more: we want to optimize the space usage as well.

The algorithm by [43] broadly works as follows for insert-only streams. It partitions the vertex set into a number of clusters and stores only intra-cluster edges during stream processing. In the post-processing phase, it colors each cluster using an offline  $(\Delta + 1)$ -coloring algorithm with pairwise disjoint palettes for the different clusters. This attains a desired  $(1 + \varepsilon)\Delta$ -coloring of the entire graph. We observe that instead, we can color each cluster on the fly using a dynamic  $(\Delta + 1)$ -coloring algorithm such as the one by [102] that takes  $O(1)$  amortized update time for maintaining a coloring. A stream update causes an edge insertion in at most one cluster and hence, the update time is the same as that required for a single run of [102]. The [43] algorithm runs roughly  $O(\log n)$  parallel sketches, and hence, we can maintain a  $(1 + \varepsilon)\Delta$ -coloring of the graph in  $\tilde{O}(1)$  update time while using the same space as [43], which is  $\tilde{O}(\varepsilon^{-2}n)$ . This proves Fact 3.2.2.

If we use this algorithm as the blackbox algorithm  $A$  in our adversarially robust algorithm for  $O(\Delta^k)$ -coloring in insert-only streams, we get  $\tilde{O}(1)$  amortized update time for each parallel copy of  $A$ , implying an  $\tilde{O}(s)$  amortized update time in total, where  $s$  is the number of parallel sketches used. We, however, also need to process a buffer deterministically, where we cannot use the aforementioned algorithm since it's randomized. We can use the deterministic dynamic  $(\Delta + 1)$ -coloring algorithm by [46] for this

part to get an additional  $\tilde{O}(1)$  amortized update time. Thus, overall, our update time is  $\tilde{O}(s) = \tilde{O}((m/n)^{1/k}) = \tilde{O}(\Delta^{1/k})$ . Finally, we can think of the algorithm as maintaining an  $n$ -length vector representing the coloring and making changes to its entries with every update while spending  $\tilde{O}(\Delta^{1/k})$  time in the amortized sense. Hence, there's no additional time required to answer queries. This is a significant improvement over a query time of  $\tilde{O}(n\sqrt{\Delta})$  or  $\tilde{O}(n)$ .

**Removing the Assumption of Prior Knowledge of  $m$ .** Observe that in Algorithm 4 as well as the algorithm described in Theorem 3.2.16, we assume that a value  $m$ , an upper bound on the number of edges, is given to us in advance. Without it, we do not know how many sketches to initialize at the start of the stream. A typical guessing trick does not seem to work since even the last sketch needs to process the entire graph and cannot be started “on the fly” if we follow our framework. In this context, we note the following. First, knowledge of an upper bound on the number of edges is a reasonable assumption, especially for turnstile streams, since an algorithm typically knows how large of an input stream it can handle. Second, for insert-only streams, we can always set  $m = n\Delta/2$  if an upper bound  $\Delta$  on the max-degree of the final graph is known; a knowledge of such a bound is reasonable since  $f(\Delta)$ -coloring is usually studied with a focus on bounded-degree graphs. Third, we can remove the assumption of knowing either  $m$  or  $\Delta$  for insert-only streams at the cost of a factor of  $\Delta$  in the number of colors and an additive  $\tilde{O}(n)$  factor in space, which we outline next.

At the beginning of the stream, we initialize  $\lfloor \log n \rfloor$  copies of the oblivious  $O(\Delta)$ -coloring semi-streaming algorithm  $A$  for the checkpoints where  $\Delta$  first attains values of the form  $2^i$  for some  $i \in [\lfloor \log n \rfloor]$ . For each  $i$ , the substream between the checkpoints with  $\Delta = 2^i$  and  $\Delta = 2^{i+1}$  can be handled using our algorithm as a blackbox since we know that the stream length is at most  $2^{i+1}n$ . This way, we need not initialize  $O(D^{1/k})$  sketches for  $D \gg \Delta_{\max}$  at the very beginning of the stream, where  $\Delta_{\max}$  is the final max-degree

of the graph, and incur such a huge factor in space; we can initialize the  $d^{1/k}$  sketches for the substream with  $d \leq \Delta \leq 2d$  only when (if at all)  $\Delta$  reaches the value  $d$ . Thus, the maximum space used is  $O(n\Delta_{\max}^{1/k})$ , which we can afford. When queried in a substream between checkpoints at  $\Delta = 2^i$  and  $\Delta = 2^{i+1}$ , we use our  $O(\Delta^k)$ -coloring algorithm to get a coloring of the substream, and take product with the  $O(\Delta)$ -coloring at the checkpoint at  $\Delta = 2^i$ . Thus, we get an  $O(\Delta^{k+1})$ -coloring of the current graph. The additional space usage is  $\tilde{O}(n)$  due to the initial  $\lfloor \log n \rfloor$  sketches taking up  $\tilde{O}(n)$  space each; hence, the total space usage is still  $O(n\Delta_{\max}^{1/k})$ .

---

## Chapter 4

---

# Streaming Verification

Interactive proof systems have contributed a very important conceptual message to computer science: it is possible for a computationally bounded entity to reduce its computational cost for a problem if it is only required to verify a proof of the solution instead of finding a solution on its own. This concept led to celebrated results such as  $IP = PSPACE$  [158] and the PCP Theorems [13, 14]. It is natural to incorporate this idea to deal with challenging problems in massive data streams so as to reduce the impractical computational costs for such problems. This incorporation led to the following setting: a space-restricted client reading a huge data stream outsources the computation to a more powerful entity, such as a cloud service, with unbounded space. The cloud sends the result of the computation to the client who refuses to blindly trust it since it might be malicious or might have incurred some hardware failure. Therefore, the cloud (henceforth named “Prover”) also sends the client (henceforth named “Verifier”) a *proof* in support of its results. Verifier needs to use his limited space to collect sufficient information from the stream so as to verify the proof. In the case that Prover is honest, Verifier can use it as a help message to find the solution to the underlying problem. Otherwise, he rejects the proof. This combination of data streaming with prover-verifier systems has been fruitful: multiple works [2, 54, 57–59, 62, 71, 73, 122, 123, 164] have shown that several intractable problems

in the basic data streaming model turn out to be solvable in prover-enhanced models using verification space and proof-length sublinear in the input size.

Past work has considered a few different instances of this setup, such as (a) *annotated* data streaming algorithms [57]—also called *online schemes*—where the parties read  $\sigma$  together and the protocol consists of Prover streaming a “help message” (a.k.a. proof) to Verifier either during stream processing and/or at the end; (b) *prescient schemes* [54, 57], which are a variant of the above where Prover knows all of  $\sigma$  before Verifier sees it; (c) *streaming interactive proofs* (SIPs) [58, 73], where Verifier and Prover engage in multiple rounds of communication.

This work focuses on the first and arguably best-motivated of these models, namely, online schemes. We simply call them *schemes*.

## Section 4.1

### Preliminaries, Setup, and Terminology

In this work, the input graph, multigraph, or digraph is denoted  $G$  and defined on a fixed vertex set  $V = [n]$ . In the *vanilla* streaming model,  $G$  is given as a stream of  $(u, v)$  tokens, where  $u, v \in V$ : the token is interpreted as an insertion of edge  $\{u, v\}$  or directed edge  $(u, v)$ . If  $G$  is edge-weighted, the tokens are of the form  $(u, v, w)$ , where  $w \in \mathbb{Z}^+$  is a weight. In the *turnstile* streaming model, tokens are of the form  $(u, v, \Delta)$ , denoting that the quantity  $\Delta \in \mathbb{Z}$  (which can be negative) is added either to the multiplicity or the weight of the edge  $\{u, v\}$ .

Throughout this paper, the stream elements come from the universe  $[n] := \{1, \dots, n\}$  and the stream length is  $m$ . In the *turnstile* streaming model, tokens are of the form  $(j, \Delta) \in [n] \times \mathbb{Z}$ , which means  $\Delta$  copies of the element  $j$  are inserted (resp. deleted) if  $\Delta > 0$  (resp.  $\Delta < 0$ ). The *cash register* or *insert-only* streaming model is the special case when  $\Delta$  is always positive. In this paper, for simplicity, we assume unit updates, i.e.,  $\Delta \in \{-1, 1\}$ .

always. The assumption can be easily removed by looking at an update as a collection of multiple unit updates.

For a stream  $\sigma = \langle (a_1, \Delta_1), \dots, (a_m, \Delta_m) \rangle$ , the *frequency vector*  $\mathbf{f}(\sigma)$  is defined as  $\langle f_1, \dots, f_n \rangle$  where  $f_j$  is the *frequency* of element  $j$ , given by  $f_j := \sum_{\substack{i \in [m]: \\ a_i = j}} \Delta_i$ . We denote *estimates* of  $f_j$  by  $\hat{f}_j$ . We drop the argument  $\sigma$  when the stream is clear from the context.

In our schemes, we use the standard technique of *sketching* a frequency vector by evaluating its *low-degree extension* at a random point. We explain what this means. We transform (or *shape*) our frequency vector of length  $n$  into a 2-dimensional  $d_1 \times d_2$  array  $f$ , where  $d_1 d_2 = n$ , using some canonical bijection from  $[n]$  to  $[d_1] \times [d_2]$ . This means that the domain of the function  $f$  can now be seen as  $[d_1] \times [d_2]$ . We work on a finite field  $\mathbb{F}$  with large enough characteristic such that the values don't "wrap around" under operations in  $\mathbb{F}$ . By Lagrange's interpolation, there is a unique polynomial  $\tilde{f}(X, Y) \in \mathbb{F}[X, Y]$  with  $\deg_X(\tilde{f}) = d_1 - 1$  and  $\deg_Y(\tilde{f}) = d_2 - 1$  such that  $\tilde{f}(x, y) = f(x, y)$  for all  $(x, y) \in [d_1] \times [d_2]$ . We call  $\tilde{f}$  the *low-degree  $\mathbb{F}$ -extension* of  $f$ . For each  $(x, y) \in [d_1] \times [d_2]$ , we have "Lagrange basis polynomials" defined as

$$\delta_{x,y}(X, Y) := \left( \prod_{x_i \in [d_1] \setminus \{x\}} \frac{X - x_i}{x - x_i} \right) \cdot \left( \prod_{y_i \in [d_2] \setminus \{y\}} \frac{Y - y_i}{y - y_i} \right) \quad (4.1)$$

We can write  $\tilde{f}$  as a linear combination of these polynomials as follows:

$$\tilde{f}(X, Y) = \sum_{(x,y) \in [d_1] \times [d_2]} f(x, y) \delta_{x,y}(X, Y)$$

In particular, if  $f$  is built up from a stream of turnstile updates  $\langle ((x, y)_j, \Delta_j) \rangle$ , then

$$\tilde{f}(X, Y) = \sum_j \Delta_j \delta_{(x,y)_j}(X, Y). \quad (4.2)$$

Thus, we can use eq. (4.2) to *maintain*  $\tilde{f}(x^*, y^*)$  at some fixed point  $(x^*, y^*)$  dynamically

with stream updates. We formalize this in the following fact.

**Fact 4.1.1.** *Given  $\mathbf{p} = (p_1, \dots, p_k) \in \mathbb{F}^k$  and a stream of pointwise updates to an initially-zero array with dimensions  $(s_1, \dots, s_k)$ , we can maintain the evaluation  $\tilde{f}(\mathbf{p})$  using  $O(\log |\mathbb{F}|)$  space, performing  $O(k)$  field arithmetic operations per update. In applications, we usually take  $\mathbf{p} \in_R \mathbb{F}^k$ .<sup>1</sup> For details and implementation considerations, see Cormode et al. [73].*  $\square$

Another useful primitive is *fingerprinting*, used prominently in our SSSP scheme and subtly in subroutines within other schemes. Its goal is to check equality between two vectors  $\mathbf{a} = (a_1, \dots, a_N)$  and  $\mathbf{b} = (b_1, \dots, b_N)$  that are provided via turnstile streams in some possibly intermixed order. This is achieved by checking that  $\varphi_{\mathbf{a}}(r) = \varphi_{\mathbf{b}}(r)$  for  $r \in_R \mathbb{F}$ , where  $\varphi_{\mathbf{a}}(X) = \sum_{j=1}^N a_j X^j$  is the *fingerprint polynomial* of  $\mathbf{a}$  and has degree at most  $N$ . Both fingerprinting and the eventual uses of Fact 4.1.1 in sum-check protocols depend upon the following basic but powerful result.

**Fact 4.1.2** (Schwartz–Zippel Lemma). *For a nonzero polynomial  $P(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$  of total degree  $d$ , where  $\mathbb{F}$  is a finite field,  $\Pr_{(r_1, \dots, r_n) \in_R \mathbb{F}^n} [P(r_1, \dots, r_n) = 0] \leq d/|\mathbb{F}|$ .*  $\square$

At various points, we shall use a couple of schemes from Chakrabarti et al. [54, 57].

**Fact 4.1.3** (SUBSET and INTERSECTION schemes; Prop. 4.1 of [57] and Thm. 5.3 of [54]). *Given a stream of elements of sets  $S, T \subseteq [N]$  interleaved arbitrarily, for any  $h, v$  with  $hv \geq N$ , there are  $[h, v]$ -schemes to compute  $|S \cap T|$  and to determine whether  $S \subseteq T$ .*  $\square$

**Setup and Terminology.** We formalize the setting described above. A *scheme* for computing a function  $g(\sigma)$  of the input stream  $\sigma$  is a triple  $(\mathcal{H}, \mathcal{A}, \text{out})$ , where  $\mathcal{H}$  is a function that Prover uses to generate the help message or proof-stream for  $\sigma$ , given by  $\mathcal{H}(\sigma)$ ,  $\mathcal{A}$  is a data streaming algorithm that Verifier runs on the stream  $\sigma$  using a random string  $R$

<sup>1</sup>The notation  $r \in_R A$  means that  $r$  is drawn uniformly at random from the finite set  $A$ .

to produce a summary  $\mathcal{A}_R(\sigma)$ , and out is a streaming algorithm that Verifier runs on the proof-stream  $\mathcal{H}(\sigma)$  and also uses  $\mathcal{A}_R(\sigma)$  and  $R$  to generate an output  $\text{out}_R(\mathcal{H}(\sigma), \mathcal{A}_R(\sigma))$  in  $\text{range}(g) \cup \perp$ , where the symbol  $\perp$  denotes rejection of the proof. Note that if the proof-length  $|\mathcal{H}(\sigma)|$  is larger than the memory of Verifier, then he needs to process  $\mathcal{H}(\sigma)$  as a stream.

A scheme  $(\mathcal{H}, \mathcal{A}, \text{out})$  has completeness error  $\varepsilon_c$  and soundness error  $\varepsilon_s$  if it satisfies

- (completeness)  $\forall \sigma : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), \mathcal{H}(\sigma)) = g(\sigma)] \geq 1 - \varepsilon_c$ ;
- (soundness)  $\forall \sigma, H : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), H) \notin \{g(\sigma), \perp\}] \leq \varepsilon_s$ .

Informally, this means that an honest Prover can convince Verifier to produce the correct output with high probability. Again, if Prover is dishonest, then, with high probability, Verifier rejects the proof. We usually aim for  $\varepsilon_c, \varepsilon_s \leq 1/3$  (they can be boosted down using standard techniques incurring a small increase in the space usage). A scheme is said to have *perfect completeness* if  $\varepsilon_c = 0$ .

The *hcost* (short for “help cost”) of a scheme  $(\mathcal{H}, \mathcal{A}, \text{out})$  is defined as  $\max_{\sigma} |\mathcal{H}(\sigma)|$ , i.e., the maximum number of bits required to express a proof. The *vcost* (short for “verification cost”) is the maximum bits of space used by the algorithms  $\mathcal{A}_R(\sigma)$  and  $\text{out}_R(\sigma)$ , where the maximum is taken over all inputs  $\sigma$  and possible random strings  $R$ . A scheme with *hcost*  $O(h)$  and *vcost*  $O(v)$  is called an  $(h, v)$ -scheme. An  $(h, v)$ -scheme is interesting if  $h > 0$  and  $v$  is asymptotically smaller than the best bound achievable for  $h = 0$ , i.e., in the basic (sans prover) streaming model.

## Section 4.2

### Frequency-Based Functions

Since its inception, data streaming algorithms have been extensively studied for fundamental statistical problems such as counting the number of distinct elements in a stream

( $F_0$ ) [10, 25, 83, 114], the  $k$ th frequency moment for  $k > 0$  ( $F_k$ ) [10, 86, 105, 168], the maximum frequency of an element ( $F_\infty$ ) [10, 113], and the  $\ell_p$ -norm of the frequency vector for some  $p \geq 0$  [113, 114, 144]. All of these problems are special cases of (or can be solved by easily applying) the general problem of computing *frequency-based functions*: given a function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ , find  $\sum_{j=1}^n g(f_j)$ , where, for each  $j$  in the universe  $\{1, \dots, n\}$ ,  $f_j$  is the frequency of the  $j$ th element. This general problem was notably addressed by the celebrated seminal paper by Alon, Matias, and Szegedy [10]: they asked for a characterization of precisely which frequency-based functions can be approximated efficiently in the basic streaming model. The aforementioned paper by Chakrabarti et al. [57] studied such statistical problems in the annotated streaming setting and gave several interesting schemes. In particular, for the general problem of computing frequency-based functions, they gave an  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -scheme. Their scheme uses an intricate data structure with binary trees and calls upon a subroutine for heavy-hitters that uses an elaborate framework called *hierarchical heavy hitters*.

Given how general the problem is, with several important special cases having numerous applications, it is important and beneficial to have a *simple* scheme for the general problem. In this work, we design such a simple scheme that uses the most basic and classical data structure for frequency estimation: the Misra-Gries summary [141]. Our scheme ends up improving the best known complexity bounds for the problem: we give an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme. No better bounds or simpler algorithms were known even for the special cases of computing  $F_0$  or  $F_\infty$ . Our result thus simplifies and improves the bounds for these problems as well.

The aforementioned scheme works for streams of length  $m = O(n)$ , an assumption that was also made by Chakrabarti et al. [57]. However, their scheme can be made to work for longer turnstile streams as long as  $\|f\|_1 = O(n)$ . We show how to use the Count-Median Sketch [72], an estimation algorithm with stronger guarantees than Misra-Gries, to

get a scheme with similar complexity bounds for these long streams. But since the Count-Median Sketch is randomized (contrary to Misra-Gries), we incur a non-zero completeness error for this scheme. The high-level idea of both our schemes is the following: we use the estimation algorithm as a primitive to “partially” solve the problem. Prover then helps Verifier with the “remaining” unsolved part.

#### 4.2.1. Our Results and Techniques

In this section, we state our results and give an overview of our techniques.

**Results.** Given a stream with elements in  $[n]$ , let  $\mathbf{f}$  denote its frequency vector  $\langle f_1, f_2, \dots, f_n \rangle$ , where  $f_j$  is the frequency of the  $j$ th element. A *frequency-based function* is a function  $G(\mathbf{f})$  of the form  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  for some function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ .

Our main result is captured in the following theorem which we prove in Section 4.2.3.

**Theorem 4.2.1.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream of length  $m = O(n)$ . The scheme is perfectly complete and has soundness error at most  $1/\text{poly}(n)$ .*

With some modifications, we obtain a similar scheme for longer streams at the cost of imperfect completeness. This is given by the following theorem which we prove in Section 4.2.4.

**Theorem 4.2.2.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ . The scheme has completeness and soundness errors at most  $1/3$ .*

As a consequence, we get schemes with the same complexity bounds for the problems of computing  $F_0$ ,  $F_\infty$ , and checking multiset inclusion (see Corollary 4.2.5 for formal definition). Just as for frequency-based functions, our schemes also improve upon the best

known bounds for these special cases and applications<sup>2</sup>. We discuss these results in detail in Section 4.2.4.

**Corollary 4.2.3.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_0$ , the number of distinct elements with non-zero frequency, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

**Corollary 4.2.4.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_\infty$ , the maximum frequency of an element, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

**Corollary 4.2.5.** *Let  $X, Y \subseteq [n]$  be multisets of size  $O(n)$ . Given a stream where elements of  $X$  and  $Y$  arrive in interleaved manner, there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for determining whether  $X \subseteq Y$ .*

**Techniques.** Computing frequency-based functions is challenging simply because we don't have enough space to store all the exact frequencies. However, there are efficient small-space algorithms—e.g., Misra-Gries algorithm [141], Count-Median Sketch [72]—that return reasonably good *estimates* of the frequencies. We use such an algorithm as a primitive in our schemes. The estimates returned partially solve the problem by helping us identify the “heavy-hitters” or the most frequent items. There cannot be too many heavy-hitters and hence, the all-powerful Prover can send Verifier the exact frequencies of these elements (which of course need to be verified) without too much communication. On the other hand, the rest of the elements, though large in number, have relatively small frequency. We show a way to encode the answer in terms of a low-degree polynomial

---

<sup>2</sup>Computing  $F_k$  for constant  $k > 0$  is a well-studied special case for which better bounds are known [57].

when the frequencies are small. Prover can then send us this polynomial using few bits, enabling us to solve the problem with small communication overall.

We remark that the high-level technique used in our first scheme—using Misra-Gries as a subroutine—might be more widely applicable than that used in the second one, i.e., using Count-Median Sketch. This is because Misra-Gries is deterministic while Count-Median is randomized. In general, both Prover and Verifier can locally run a *deterministic* algorithm on the input, and then, Prover can send messages based on the final state of that algorithm. Note that it isn't clear if a *randomized* algorithm can always help in this regard since we assume that Prover and Verifier do not have access to shared randomness<sup>3</sup>. Hence, the final states of the algorithm might vary drastically for Prover and Verifier if they run it locally with their own private randomness. For our problem, we don't run into this issue since we don't require Prover to know the exact output of the Verifier's local estimation algorithm.

Other techniques used are pretty standard in this area. We use techniques based on the famous *sum-check protocol* of Lund et al. [135] that encodes answers as sum of low-degree polynomials. In our case, where Prover sends only a single message to Verifier, a quantity of interest is expressed as the sum of evaluations of a low-degree univariate polynomial. Since the polynomial has low-degree, it can be expressed with a small number of monomials. Thus, Prover needs only a few bits to express the set of coefficients that describe the polynomial, leading to short proof-length. Moreover, to verify the authenticity of the polynomial, Verifier needs to evaluate it at just a single random point, the space for which he can afford. The main challenge in this technique is to find the proper low-degree polynomials to encode the answer, and in this work, we give such new polynomial encodings for the underlying sub-problems. Another standard technique we use is the *shaping technique* that transforms a one-dimensional vector into a two-dimensional array. On a high-level, this helps in “distributing” the work between Prover and Verifier as they

---

<sup>3</sup>This assumption is made so that it corresponds to the MA communication model. Access to shared randomness corresponds to the AMA communication model where better bounds are known [94].

each “take care of” a single dimension. Pertaining to the streaming model, we exploit the popular technique of *linear sketching* where we express a quantity of interest as a linear combination of the stream updates, which helps us to maintain the quantity dynamically as the stream arrives.

**Related Work.** Early works on the concept of stream outsourcing and verification were done by the database community [131, 148, 166, 172]. Motivated by these works, Chakrabarti et al. [57] abstracted out and formalised the theoretical aspects of the settings. They defined two types of stream verification settings: (i) the *annotated data streaming* setting—calling the schemes as *online schemes*—where Prover and Verifier read the input stream together and Prover sends help messages during and/or after the stream arrival based on the part of the stream she has seen so far, and (ii) the *prescient* setting where Prover knows the entire stream upfront, i.e., before Verifier sees it, and can send help messages accordingly. Several subsequent works [54, 59, 62, 71, 122, 164] studied these non-interactive models. Natural generalizations of the model, where we allow multiple rounds of interaction between Prover and Verifier, have also been explored. These include *Arthur-Merlin streaming protocols* (Prover is named “Merlin” and Verifier is named “Arthur” following a long-standing tradition in complexity theory) of Gur and Raz [94] and the *streaming interactive proofs* (SIP) of Cormode et al. [73]. The latter setting was further studied by multiple works [2, 58, 123]. We refer the reader to the expository article by Thaler [163] for a detailed survey of this area.

We state the results with the standard assumption [57, 73] that the stream length  $m = O(n)$ . Chakrabarti et al. [57] gave two schemes for computing any general frequency-based function: an online  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -scheme and a prescient  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme. They noted that the schemes apply to get best known schemes for the special cases of computing the number of distinct elements ( $F_0$ ), the maximum frequency ( $F_\infty$ ), and for checking multiset inclusion. They also showed a lower bound that any online or prescient

$(h, v)$ -scheme for the problem (even for the aforementioned special cases) requires  $hv \geq n$ . They designed schemes with better bounds for certain other frequency-based functions, often matching this lower bound up to polylogarithmic factors. For instance, for any  $hv = n$ , they gave an online  $(k^2 h \log n, kv \log n)$ -scheme for calculating the  $k$ th frequency moment  $F_k$  for any positive integer  $k$ , and a  $(\phi^{-1} \log^2 n + h \log n, v \log n)$ -scheme for computing the  $\phi$ -heavy hitters (elements with frequency of at least a  $\phi$ -fraction of the stream length).

The specific problem of computing  $F_0$  has been studied by multiple works in various stream verification models. Cormode, Mitzenmacher, and Thaler [71] studied the problem in the stronger SIP-model and gave a  $(\log^3 n, \log^2 n)$ -SIP with  $O(\log^2 n)$  rounds of communication. For the case where we restrict the number of rounds to  $O(\log n)$ , Cormode, Thaler, and Yi [73] gave a  $(\sqrt{n} \log^2 n, \log^2 n)$ -SIP. Klauck and Prakash [123] improved this to a  $(\log^4 n \log \log n, \log^2 n \log \log n)$ -SIP. Gur and Raz [94] designed an  $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$ -AMA-streaming protocol<sup>4</sup> for  $F_0$ .

#### 4.2.2. The Misra-Gries Algorithm

An important subroutine in one of our schemes is the classic Misra-Gries algorithm for frequency estimation [141] which, given an input stream of  $m$  elements and a fraction  $\phi$ , estimates the frequency of the stream elements within an additive factor of  $\phi m$ . We recall this algorithm in Algorithm 5.

Informally, the algorithm does the following: it keeps an array or “dictionary”  $K$  indexed by “keys” that are elements of the stream and each of them has an associated counter  $K[i]$ . At any point of time, the array has at most  $\lceil \phi^{-1} \rceil$  keys. When a stream element arrives, it increments the counter for the element if it’s present in the keys (it includes it in the keys if there are less than  $\lceil \phi^{-1} \rceil$  keys), and otherwise decrements the counter of every key. If a counter for a key becomes 0, it is removed from  $K$ . Finally, the estimate  $\hat{f}_j$  is given by  $K[j]$  (which is 0 if  $j$  is not in the keys). The guarantees of the algorithm is given

<sup>4</sup>AMA stands for the communication pattern Arthur-Merlin-Arthur

in Fact 4.2.1.

---

**Algorithm 5** [141] Misra-Gries algorithm for frequency estimates in insert-only streams

---

**Require:** Stream  $\sigma$ ;  $\phi \leq 1$

1: Initialize  $K \leftarrow$  empty array

**Process**(token  $j \in \sigma$ ):

2: **if**  $j \in \text{keys}(K)$  **then**

3:      $K[j] \leftarrow K[j] + 1$

4: **else**

5:     **if**  $|\text{keys}(K)| < \lceil \phi^{-1} \rceil$  **then**

6:          $K[j] \leftarrow 1$

7:     **else**

8:         **for**  $i \in \text{keys}(K)$  **do**:

9:              $K[i] \leftarrow K[i] - 1$

10:         **if**  $K[i] = 0$  **then** remove  $i$  from  $\text{keys}(K)$

**Output:**

11: **for**  $j \in [n]$  **do**:

12:     **if**  $j \in \text{keys}(K)$  **then return**  $\hat{f}_j = K[j]$ ; **else return**  $\hat{f}_j = 0$

---

**Fact 4.2.1** ([141]). *For an insert-only stream of  $m$  elements in  $[n]$ , given any  $\phi \leq 1$ , Algorithm 5 uses  $O(\phi^{-1}(\log n + \log m))$  space and returns frequency estimates  $\langle \hat{f}_j : j \in [n] \rangle$  such that, for all tokens  $j \in [n]$ , we have  $f_j - \phi m \leq \hat{f}_j \leq f_j$ .*

Note that this algorithm was designed for insert-only streams and doesn't work for turnstile streams. To use it for turnstile streams, we need to make appropriate modifications (which we do in Section 4.2.3).

### 4.2.3. Computing Frequency-based Functions in Turnstile Streams

---

Let  $\mathbf{f}$  be the frequency vector of a stream as defined in Section 4.1. Recall that a *frequency-based function* is a function  $G(\mathbf{f})$  of the form  $G(\mathbf{f}) := \sum_{j \in [n]} g(f_j)$  for some function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ . In this section, we obtain an improved  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function for some predetermined function  $g$ . As stated earlier, we design a scheme exploiting the Misra-Gries algorithm (Algorithm 5). We want

to use it as a subroutine in our problem for turnstile streams, but it works only in the insert-only model. Therefore, in Section 4.2.3, we provide a simple extension of the algorithm that attains a similar guarantee for turnstile streams. In Section 4.2.3, first, we use this extended Misra-Gries (EMG) algorithm as a subroutine for our scheme for computing frequency-based functions. Next, we show that we can instead use the Count-Median Sketch [72] to make it work for longer streams. In Section 4.2.4, we discuss some important applications of our schemes.

***Extension of Misra-Gries Algorithm for Turnstile Streams.*** The extended Misra-Gries algorithm (henceforth called “EMG algorithm”) works as follows: we process the positive and negative updates separately in two parallel copies of Algorithm 5 to estimate the total positive update and (absolute value of) the total negative update. In the second copy, we can actually think of the updates as “increments” since only negative updates are processed there. Thus, what we are actually estimating is the absolute value of the total negative update.

For each  $j$ , let the total positive update be  $f_j^+$  and (absolute value of) the total negative update  $f_j^-$ . Then, the actual frequency is  $f_j = f_j^+ - f_j^-$ . Denote the corresponding estimates given by the copies of Algorithm 5 by  $\hat{f}_j^+$  and  $\hat{f}_j^-$ . Then  $\hat{f}_j := \hat{f}_j^+ - \hat{f}_j^-$  gives a similar guarantee as Fact 4.2.1 for turnstile streams; this time, we also incur an additive error of  $\phi m$  on the upper bound.

To see this, note that by Fact 4.2.1, we have,  $\forall j \in [n]$ ,

$$f_j^+ - \phi m \leq \hat{f}_j^+ \leq f_j^+ \quad (4.3)$$

$$f_j^- - \phi m \leq \hat{f}_j^- \leq f_j^- \quad (4.4)$$

Thus, eqs. (4.3) and (4.4) give  $f_j^+ - f_j^- - \phi m \leq \hat{f}_j^+ - \hat{f}_j^- \leq f_j^+ - f_j^- + \phi m$ , i.e.,

$$f_j - \phi m \leq \hat{f}_j \leq f_j + \phi m \quad (4.5)$$

Hence, this time we get double sided error. This estimate would suffice for getting our desired scheme. Therefore, we get the following lemma.

**Lemma 4.2.6.** *Given a turnstile stream of  $m$  elements in  $[n]$ , the EMG algorithm uses  $O(\phi^{-1}(\log n + \log m))$  space and returns a summary  $\langle \hat{f}_j : j \in [n] \rangle$  such that, for all  $j \in [n]$ , we have  $f_j - \phi m \leq \hat{f}_j \leq f_j + \phi m$ .*

*Remark.* The guarantee given by the EMG algorithm may not be very useful in general for turnstile streams. This is because the total number of stream updates  $m$  can be huge, whereas the frequency of each token can be small since we allow both increments and decrements in the turnstile model. The classic Misra-Gries algorithm for insert-only model, on the other hand, has a good guarantee (Fact 4.2.1) since  $m = \|f\|_1$  in this model. However, for our purpose, the guarantee in Lemma 4.2.6 is good enough since we assume that  $m = O(n)$ .

**Schemes for Frequency-based Functions.** First, in Section 4.2.3, we describe a protocol for computing frequency-based functions in turnstile streams of length  $O(n)$  and prove Theorem 4.2.1. Next, in Section 4.2.4, we show that the scheme can be modified to work for any turnstile stream with  $\|f\|_1 = O(n)$ , proving Theorem 4.2.2. The completeness error in the latter scheme is, however, non-zero.

**Perfectly Complete Scheme for  $O(n)$ -Length Streams.** As in prior works [57, 73], we solve the problem for stream length  $m = O(n)$ . Hence, by Lemma 4.2.6, the EMG algo-

rithm takes  $O(\phi^{-1} \log n)$  space and gives, for some constant  $c$ ,

$$\forall j \in [n] : f_j - \phi cn \leq \hat{f}_j \leq f_j + \phi cn. \quad (4.6)$$

Set  $\phi = (cn^{2/3})^{-1}$ . Therefore, we have an  $O(n^{2/3} \log n)$  space algorithm that guarantees

$$\forall j \in [n] : f_j - n^{1/3} \leq \hat{f}_j \leq f_j + n^{1/3}.$$

Let  $K$  denote the set of keys in the final state of the EMG algorithm for the setting of  $\phi = 1/(cn^{2/3})$ . Observe that if  $\hat{f}_j = 0$  for some  $j$  (i.e.,  $j \notin K$ ), we know that  $f_j \in [-n^{1/3}, n^{1/3}]$ .

Define  $h(j) = \mathbb{I}\{j \notin K\}$  where  $\mathbb{I}$  is the 0-1 indicator function. We have

$$\sum_{j \in [n]} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \notin K} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \in [n]} g(f(j))h(j)$$

Let  $L := \sum_{j \in K} g(f(j))$  and  $R := \sum_{j \in [n]} g(f(j))h(j)$ . We shall compute  $L$  and  $R$  separately and add them to get the desired answer.

We *shape* (see Section 4.1) the 1D array  $[n]$  into a 2D  $n^{1/3} \times n^{2/3}$  array. Thus, we get

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} g(f(x, y))h(x, y)$$

As is standard [57], we assume that the range of the function  $g$  is upper bounded by some polynomial in  $n$ , say  $n^p$ . Pick a prime  $q$  such that  $n^{p+1} < q < 2n^{p+1}$ . We will work in the finite field  $\mathbb{F}_q$  and the upper bound on the range of  $g$  ensures that  $G(f)$  will not “wrap around” under arithmetic in  $\mathbb{F}_q$ .

Let  $\tilde{f}, \tilde{h}$  be polynomials of lowest degree over the finite field  $\mathbb{F}_q$  that agree with  $f, h$  respectively at all values in  $[n^{1/3}] \times [n^{2/3}]$ . Note that, by Lagrange’s interpolation, both  $\tilde{f}$  and  $\tilde{h}$  have degrees  $n^{1/3} - 1$  and  $n^{2/3} - 1$  in the two variables (see Section 4.1). Again, let

$\tilde{g}$  denote the polynomial of lowest degree that agrees with  $g$  at all values in  $[-n^{1/3}, n^{1/3}]$ .

Thus,  $\tilde{g}$  has degree  $2n^{1/3}$ .

Therefore, we have

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(x, y)) \tilde{h}(x, y)$$

i.e., we can write

$$R = \sum_{x \in [n^{1/3}]} P(x), \quad (4.7)$$

where the polynomial  $P$  is given by

$$P(X) = \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(X, y)) \tilde{h}(X, y) \quad (4.8)$$

To compute  $L$ , it suffices to obtain the values  $f_j$  for all  $j \in K$  since  $g$  is predetermined. In our protocol, Prover would send values  $f'_j$  that she claims to be  $f_j$  for all  $j \in K$ . Define

$$T := \sum_{j \in K} (f_j - f'_j)^2$$

Note that we have  $f_j = f'_j$  for each  $j$  if and only if  $T = 0$ . Set  $f'_j := 0$  for all  $j \notin K$ . Thus, we can rewrite  $T$  as

$$T = \sum_{j \in [n]} (f_j - f'_j)^2 (1 - h(j))$$

Using shaping as before, we get

$$T = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} (f(x, y) - f'(x, y))^2 (1 - h(x, y)) \quad (4.9)$$

Let  $\tilde{f}'$  denote the polynomial of lowest degree over  $\mathbb{F}_q$  that agrees with  $f'$  at all values in

$[n^{1/3}] \times [n^{2/3}]$ . Therefore, we have

$$T = \sum_{x \in [n^{1/3}]} Q(x) \quad (4.10)$$

where the polynomial  $Q$  is given by

$$Q(X) = \sum_{y \in [n^{2/3}]} (\tilde{f}(X, y) - \tilde{f}'(X, y))^2 (1 - \tilde{h}(X, y)). \quad (4.11)$$

We are now ready to describe the protocol.

*Stream processing.* Verifier picks  $r \in \mathbb{F}_q$  uniformly at random. As the stream arrives, he maintains  $\tilde{f}(r, y)$  for all  $y \in [n^{2/3}]$  (Fact 4.1.1). In parallel, he runs the EMG algorithm setting  $\phi = (cn^{2/3})^{-1}$ .

*Help message.* Prover sends polynomials  $P'$  and  $Q'$ , and values  $f'_j$  for all  $j \in K$ . She claims that  $P', Q', f'$  are identical to  $P, Q, f$  respectively. The polynomials are sent as streams of their coefficients following some canonical order of their monomials. Verifier evaluates  $P'(r)$  and  $Q'(r)$  as the polynomials are streamed.

*Verification and output.* Looking at the final state of the EMG subroutine, Verifier constructs  $\tilde{h}(r, y)$  for all  $y \in [n^{2/3}]$  (he can treat the keys as a stream and use Fact 4.1.1). Also, from the values  $f'_j$ , he constructs  $\tilde{f}'(r, y)$  for all  $y \in [n^{2/3}]$ . The  $O(n^{1/3})$ -degree polynomial  $\tilde{g}$  is computed and stored in advance (we need to evaluate  $g$  at all points in  $[-n^{1/3}, n^{1/3}]$  and then use Lagrange interpolation to get  $\tilde{g}$ ).

Thus, Verifier can now use eq. (4.8) to compute  $P(r)$  and eq. (4.11) to compute  $Q(r)$ . He checks whether  $P(r) = P'(r)$  and  $Q(r) = Q'(r)$ . If the checks pass, he believes  $P', Q'$  are correct. He further checks whether  $\sum_{x \in [n^{1/3}]} Q'(x) = 0$ , i.e., by Equation (4.10), whether  $T = 0$ . If so, he believes that  $f'_j = f_j$  for all  $j \in K$ . Next, he computes  $L = \sum_{j \in K} g(f'(j))$ , and using Equation (4.7), he computes  $R =$

$\sum_{x \in [n^{1/3}]} P'(x)$ . Finally,  $L + R$  gives the answer.

*Error probability.* The correctness analysis follows along standard lines of sum-check protocols. The scheme is perfectly complete since it follows from above that we always output correctly if Prover is honest. For soundness, note that the protocol fails if either  $P \neq P'$  or  $Q \neq Q'$ , but  $P(r) = P'(r)$  and  $Q(r) = Q'(r)$ . Then,  $r$  is a root of the non-zero polynomial  $P - P'$  or  $Q - Q'$ . Since degree of  $P - P'$  is  $O(n^{2/3})$  and that of  $Q - Q'$  is  $O(n^{1/3})$ , they have at most  $O(n^{2/3})$  roots in total. Since  $r$  is drawn uniformly at random from  $\mathbb{F}_q$ , where  $q > n^{p+1}$ , the probability that  $r$  is such a root is at most  $O(n^{2/3})/n^{p+1} \leq 1/\text{poly}(n)$  for sufficiently large  $n$ . Thus, the soundness error is at most  $1/\text{poly}(n)$ .

*Help and Verification costs.* The polynomials  $P$  and  $Q$  have degree  $O(n^{2/3})$  and  $O(n^{1/3})$  respectively. Thus, it requires  $O(n^{2/3} \log n)$  bits in total to express their coefficients since each coefficient comes from  $\mathbb{F}_q$  that has size  $\text{poly}(n)$ . Recall that for the setting of  $\phi = (cn^{2/3})^{-1}$ , there are  $O(\phi^{-1}) = O(n^{2/3})$  keys in the EMG algorithm. Prover sends  $f'_j$  for each  $j \in K$ , and since each frequency is at most  $m = O(n)$ , this requires  $O(n^{2/3} \log n)$  bits to communicate. Therefore, the total hcost is  $O(n^{2/3} \log n)$ .

As noted above, the invocation of EMG algorithm takes  $O(n^{2/3} \log n)$  space. Verifier maintains  $\tilde{f}(r, y)$  and stores the values  $\tilde{h}(r, y)$  and  $\tilde{f}'(r, y)$  for all  $y \in [n^{2/3}]$ . Each value is an element in  $\mathbb{F}_q$ , and hence they take up  $O(n^{2/3} \log n)$  space in total. The  $O(n^{1/3})$ -degree polynomial  $\tilde{g}$  takes  $O(n^{1/3} \log n)$  space to store. Hence, the total vcost is  $O(n^{2/3} \log n)$ .

Thus, we have proved the following theorem.

**Theorem 4.2.1.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream of length  $m = O(n)$ . The scheme is perfectly complete and has soundness error at most  $1/\text{poly}(n)$ .*

#### 4.2.4. Modifications for Longer Streams

The scheme in Section 4.2.3 requires stream length  $m = O(n)$ . Note that a turnstile stream with massive cancellations can have length  $m \gg n$ , but  $\|\mathbf{f}\|_1$  can still be  $O(n)$ . Chakrabarti et al. [57] presented their scheme under the assumption of  $m = O(n)$ , but their scheme can be made to work for longer streams as long as  $\|\mathbf{f}\|_1 = O(n)$ . We can modify our scheme to handle such streams as well without increasing the costs, but we no longer have perfect completeness. We give a sketch of this scheme below highlighting the modifications.

We cannot use the EMG algorithm anymore because it doesn't give a strong guarantee with respect to  $\|\mathbf{f}\|_1$  for turnstile streams. We use the Count-Median Sketch instead which gives the following guarantee.

**Fact 4.2.2** (Count-Median Sketch [72]). *For all  $\phi, \varepsilon > 0$ , there exists an algorithm that, given a turnstile stream of elements in  $[n]$  with  $\|\mathbf{f}\|_1 = O(n)$ , uses  $O(\phi^{-1} \log(\varepsilon^{-1}) \log n)$  space and returns frequency estimates  $\langle \hat{f}_j : j \in [n] \rangle$  such that, with probability at least  $1 - \varepsilon$ , for all tokens  $j \in [n]$ , we have  $f_j - \phi \|\mathbf{f}\|_1 \leq \hat{f}_j \leq f_j + \phi \|\mathbf{f}\|_1$ .*

If  $\|\mathbf{f}\|_1 \leq cn$  for some constant  $c$ , then setting  $\phi = (4cn^{2/3})^{-1}$  and  $\varepsilon = 1/4$ , we get that there is an  $O(n^{2/3} \log n)$  space algorithm that, with probability at least  $3/4$ , gives

$$\forall j \in [n] : f_j - n^{1/3}/4 \leq \hat{f}_j \leq f_j + n^{1/3}/4 \quad (4.12)$$

For this protocol, redefine the set  $K$  as  $K := \{j : |f_j| \geq n^{1/3}/2\}$ . Prover sends a set  $K'$  that she claims is identical to  $K$ . Let  $M$  denote the set  $\{j : |\hat{f}_j| \geq 3n^{1/3}/4\}$ . Verifier checks whether  $M \subseteq K'$ , and if the check passes, he computes  $\sum_{j \in K'} g(f_j)$  and  $\sum_{j \notin K'} g(f_j)$  separately, similar to the earlier protocol, and adds them to obtain the answer.

**Error probability.** For completeness, note that if Prover is honest and  $K' = K$ , then with probability at least  $3/4$ , we have  $M \subseteq K'$ . To see this, observe that, by the guarantees of the Count-Median Sketch (eq. (4.12)), for all  $j \in [n]$  with  $|\hat{f}_j| \geq 3n^{1/3}/4$ , we have

$|f_j| \geq n^{1/3}/2$  with probability at least  $3/4$ . The rest of the completeness analysis is as before, and hence, there is no additional completeness error. Thus, the total completeness error of the scheme is at most  $1/4$ .

For soundness, suppose that  $K' \neq K$ . By the guarantees of the Count-Median Sketch, for all  $j \in [n]$  with  $|f_j| \geq n^{1/3}$ , we have  $|\hat{f}_j| \geq 3n^{1/3}/4$  with probability at least  $3/4$ . Thus,  $\{j : |f_j| \geq n^{1/3}\} \subseteq M$ . Hence, if the check  $M \subseteq K'$  passes, then with probability at least  $3/4$ , we have  $\{j : |f_j| \geq n^{1/3}\} \subseteq K'$ . Thus, if  $j \notin K'$ , we have  $|f_j| < n^{1/3}$ . Therefore, the computation of  $\sum_{j \notin K'} g(f_j)$  goes through as before. The additional soundness error is at most  $1/\text{poly}(n)$  as analyzed earlier. Thus, the total soundness error of the protocol is at most  $1/4 + 1/\text{poly}(n) < 1/3$ .

**Help and Verification costs.** Clearly, since  $\|\mathbf{f}\|_1 \leq cn$ , we have  $|K| = O(n^{2/3})$  which adds  $O(n^{2/3} \log n)$  bits to the hcost. The Count-Median Sketch takes space  $O(n^{2/3} \log n)$ , similar to the EMG algorithm. The rest of the cost analysis is as before, and hence we have an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme.

Thus, we have the following theorem.

**Theorem 4.2.2.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ . The scheme has completeness and soundness errors at most  $1/3$ .*

*Remark.* We compare the schemes for Theorem 4.2.1 and Theorem 4.2.2 (call them Scheme 1 and Scheme 2 respectively). Scheme 2 works for streams of length  $m \gg n$  as long as  $\|\mathbf{f}\|_1 = O(n)$ , while Scheme 1 requires  $m = O(n)$ . On the negative side, Scheme 2 has imperfect completeness, contrary to Scheme 1. Furthermore, the space dependence on the error  $\varepsilon$  for Scheme 2 is worse than Scheme 1: given any  $\varepsilon$ , Scheme 2 uses  $O(n^{2/3} \log n \log(\varepsilon^{-1}))$  space to bound the completeness and soundness errors by at most  $\varepsilon$ , while Scheme 1 takes  $O(n^{2/3}(\log n + \log(\varepsilon^{-1})))$  space to bound the soundness error by  $\varepsilon$ . This means that to bound the error by  $1/\text{poly}(n)$ , Scheme 2 takes  $O(n^{2/3} \log^2 n)$  space, making it weaker

(though simpler) than the scheme of Chakrabarti et al. [57], which takes  $O(n^{2/3} \log^{4/3} n)$  space for the same and is also perfectly complete. For this, Scheme 1 takes only  $O(n^{2/3} \log n)$  space.

**Special Instances and Applications.** Here, we note important implications of Theorems 4.2.1 and 4.2.2. They can be applied to get similar results for multiple well-studied problems such as computing the number of distinct elements in the stream ( $F_0$ ), the highest frequency of an element in the stream ( $F_\infty$ ), and checking multiset inclusions. Note that for these problems, to the best of our knowledge, the best-known schemes were  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -schemes obtained by direct application of the general scheme. Hence, we improve the bounds and simplify the schemes for these problems as well.

As a direct corollary of Theorems 4.2.1 and 4.2.2, we get the same bounds for  $F_0$ . It is an extensively studied problem in both basic streaming and stream verification. It is the special case of frequency-based functions where the function  $g$  is defined as  $g(x) = 0$  if  $x = 0$ , and  $g(x) = 1$  otherwise. Therefore, we obtain the following result.

**Corollary 4.2.3.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_0$ , the number of distinct elements with non-zero frequency, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

Another well-studied problem related to frequency-based functions is computing  $F_\infty$ . Unlike  $F_0$ , it is not a direct special case, but a protocol for it follows by easily applying a scheme for frequency-based functions. Chakrabarti et al. [57] noted one way in which it can be applied to solve  $F_\infty$ . Here, we note a slightly alternate way which doesn't use a subroutine that their scheme uses and is tailored to our protocols: Prover sends the element  $j^* \in [n]$  that she claims has the highest frequency and a value  $f'_{j^*}$  that she claims to be equal to  $f_{j^*}$ . By the above protocols, Verifier can check whether  $f'_{j^*} = f_{j^*}$ . If the check

passes, he computes  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  using the scheme above, where  $g$  is defined as  $g(x) = 0$  if  $x \leq f'_{j*}$  and  $g(x) = 1$  otherwise. He accepts Prover's claim if  $G(\mathbf{f}) = 0$ . Thus, we get the following result.

**Corollary 4.2.4.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_\infty$ , the maximum frequency of an element, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

The problem of checking multiset inclusion has two multisets arriving in a stream arbitrarily interleaved between each other, and we need to check if one of them is contained in the other. This abstract problem is used as a subroutine in several other problems, e.g., some graph problems considered in the annotated settings [57, 59, 62]. Thus, an improved scheme for multiset inclusion implies improved subroutines for the corresponding problems. It can be solved by easy application of frequency-based functions. The reduction is already noted in Chakrabarti et al. [57], but we repeat it here for the sake of completeness.

**Corollary 4.2.5.** *Let  $X, Y \subseteq [n]$  be multisets of size  $O(n)$ . Given a stream where elements of  $X$  and  $Y$  arrive in interleaved manner, there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for determining whether  $X \subseteq Y$ .*

*Proof.* Think of  $X$  and  $Y$  as  $n$ -length characteristic vector representation of the multisets (with an entry denoting the multiplicity of the corresponding element). Then,  $X \subseteq Y$  if and only if  $X_j \leq Y_j$  for each  $j \in [n]$ . As the elements arrive, we increment an entry if belongs to  $Y$  and decrement it if it belongs to  $X$ . Thus, the vector  $\mathbf{f}$  is given by  $f_j = Y_j - X_j$ . Define  $g$  as  $g(x) = 0$  if  $x \geq 0$  and  $g(x) = 1$  otherwise. Therefore, computing  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  and checking if it equals 0 solves the problem. The multisets having size  $O(n)$  ensures that the length of the stream is  $O(n)$ , and so we can safely apply our scheme.  $\square$

## Section 4.3

## Graph Problems

Several recent works in the annotated stream and the SIP models have focused on basic algorithmic problems on graphs [2, 71, 164], often giving sublinear-space algorithms for problems that provably do not admit sublinear solutions in the basic (sans prover) streaming setting.

In this work, we give new algorithms in the annotated streaming setting for certain graph problems, including triangle counting, its generalization to subgraph counting, maximum matching, problems about the existence (or not) of short paths, finding the shortest path between two vertices, and testing for an independent set. Two of our results provide “unexpected” new upper bounds, disproving published conjectures [164] asserting that such bounds would be unattainable. We give new and improved schemes for several graph-theoretic problems, including triangle counting, maximum matching, topological sorting, and shortest paths. In all cases, the input is a huge  $n$ -vertex graph  $G$  given as a stream  $\sigma$  of edge insertions and/or deletions. While most of our problems have been studied before, we give schemes that (a) have better complexity parameters, in some cases achieving optimality, and (b) use cleverer algebraic encodings of the relevant combinatorial problems, often exploiting the ability of a streaming algorithm to compute *nonlinear* sketches.

**Subgraph Counting.** The literature on graph streaming contains many works on the central problem of triangle counting (henceforth, TRIANGLECOUNT): given a multigraph  $G$  as a dynamic stream, compute  $T$ , the number of triangles in  $G$  [26, 42, 106, 139, 164]. In Section 4.3.3, we study this and the more general problem of subgraph-counting (SUBGRAPHCOUNT $_k$ ) [42, 111, 112, 164], where the goal is to compute  $T_H$ , the number of copies of a fixed,  $k$ -sized graph  $H$ , where  $k$  is a constant. In the basic streaming model, computing  $T$  or  $T_H$  exactly is impossible in sublinear space and it becomes necessary to approximate. In con-

trast, we design a family of  $(o(n^2), o(n))$ -schemes for TRIANGLECOUNT that give exact answers. Such a frugal scheme had been conjectured not to exist [164]. We extend our ideas to give sublinear  $(o(n^2), o(n^2))$ -schemes for SUBGRAPHCOUNT<sub>k</sub>.

**Maximum Matching.** Determining  $\alpha'(G)$ , the cardinality of a maximum-sized matching in  $G$ , is a central problem in graph algorithms and has received a lot of attention in the recent literature on streaming algorithms [18, 63, 82, 91, 116, 136]. In Section 4.3.5, we consider this problem (henceforth, MAXMATCHING) for multigraphs given by dynamic streams. As with TRIANGLECOUNT, we give a frugal scheme for MAXMATCHING, which had been conjectured to be impossible [164]. In the process, we present a frugal scheme for the subproblem of verifying that the purported connected components of a graph are indeed disconnected from each other, which might be of independent interest for future work on connectivity-related problems.

**Independent Sets and Length-Three Paths.** In Section 4.4, we study the independent set testing problem (INDSETTEST), where we are given a multigraph  $G$  and a set  $U \subseteq V$  (also streamed and interleaved with the edge stream arbitrarily) and we must determine whether or not  $U$  is independent. We also study the ST-3PATH problem, where  $G$  (which might be a digraph) has two designated vertices  $v_s$  and  $v_t$  and we must determine whether  $G$  has a path of length at most 3 from  $v_s$  to  $v_t$ . By results from prior work, any  $(h, v)$ -scheme for these problems must have total cost  $h + v = \Omega(n)$ . We therefore design two-pass schemes for these problems, achieving  $h + v = \tilde{O}(n^{2/3})$ . In fact, we obtain a more general tradeoff, giving a two-pass  $[t^2, s]$ -scheme for any parameters  $t, s$  with  $ts = n$ . Our schemes instantiate a protocol for the abstract problem CROSSEDGECOUNT, which asks for a count of the number of edges in  $G$  from  $U \subseteq V$  to  $W \subseteq V$ , where these sets  $U$  and  $W$  are also streamed.

In each case, we *can* design ordinary (one-pass) schemes with the same complexity parameters under a natural assumption on the way the stream is ordered, and these schemes

still beat the space bound achievable by basic (sans prover) streaming algorithms.

**Short Paths and Shortest Path.** Finally, in Section 4.3.7, we consider shortest path problems, perhaps the most basic problem in classic graph algorithms. We study the ST-KPATH problem, which is to detect whether or not  $G$  has a path of length at most  $k$  from  $v_s$  to  $v_t$ , where  $k$ ,  $v_s$ , and  $v_t$  are prespecified. We first present a  $[kn, n]$ -scheme for ST-KPATH. This gives a semi-streaming scheme for detecting short (of length polylogarithmic in  $n$ ) paths, which is optimal in terms of total cost. It also implies a  $[kn, n]$ -scheme for ST-SHORTESTPATH problem—where  $k$  is the length of the shortest path from  $v_s$  to  $v_t$ —which is to find the shortest path between vertices  $v_s$  and  $v_t$ , and output NO if none exists. For directed graphs of small (polylogarithmic in  $n$ ) diameter, it implies a semi-streaming scheme for checking  $v_s$ – $v_t$  connectivity. Note that these problems require  $\Omega(n^2)$  space in the basic data streaming model, even for constant  $k$  or constant-diameter graphs [82].

Targeting a different cost regime, we generalize our result for ST-3PATH from Section 4.4 to obtain multi-pass  $(h, v)$ -schemes for ST-KPATH with total cost  $h + v = o(n)$ , for constant  $k$ . To be precise, we present a  $\lceil k/2 \rceil$ -pass  $[n^{1-1/k}, n^{1-1/k}]$ -scheme for ST-KPATH.

#### 4.3.1. Our Techniques

**Sum-Check and Polynomial Encodings.** As with much prior work in this area (and probabilistic proof systems more generally), our schemes are variants of the famous *sum-check protocol* of Lund et al. [135]. Specialized to our (non-interactive) schemes, this protocol allows Verifier to make Prover honestly compute  $\sum_{x \in \mathcal{X}} g(x)$  for some low-degree polynomial  $g(X)$  derived from the input data and some designated set  $\mathcal{X}$ . Verifier has no space to compute  $g$  explicitly, nor all values  $\langle g(x) : x \in \mathcal{X} \rangle$ , but he can afford to evaluate  $g(r)$  at a *random* point  $r$ . The Prover steps in by explicitly providing  $\hat{g}(X)$ , a polynomial claimed to equal  $g(X)$ : this is cheap since  $g$  has low degree. Verifier can be convinced of this claim by checking that  $\hat{g}(r) = g(r)$ .

Hence, the main challenge in applying the sum-check technique is to find a way to encode the data stream problem's output as the sum of the evaluations of a low-degree polynomial  $g$  so that Verifier can, in small space, evaluate  $g$  at a random point  $r$ .

**Sketches: Linearity and Beyond.** A streaming Verifier evaluates  $g(r)$  by suitably summarizing the input in a *sketch*. Viewing the input as updates to a data vector  $\mathbf{f} = (f_1, \dots, f_N)$ , such a sketch  $\mathbf{v}$  is *linear* if  $\mathbf{v} = S\mathbf{f}$  for some matrix  $S \in \mathbb{F}^{v \times N}$ , for some field  $\mathbb{F}$ .<sup>5</sup> Typically,  $S$  is implicit in the sketching algorithm and enables stream processing in  $\tilde{O}(v)$  space by translating a stream update  $f_i \leftarrow f_i + \Delta$  into the sketch update  $\mathbf{v} \leftarrow \mathbf{v} + \Delta S \mathbf{e}_i$ , where  $\mathbf{e}_i$  is the  $i$ th standard basis vector. In essentially all prior works on stream verification, the polynomial  $g$  was such that  $g(r)$  could be derived from such a linear sketch  $\mathbf{v}$ .

There is one exception: Thaler [164] introduced an optimal  $[n, n]$ -scheme for TRIANGLECOUNT in which Verifier computes a *nonlinear* sketch.<sup>6</sup> Roughly speaking, the verifier in Thaler's protocol maintains two  $n$ -dimensional linear sketches  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$ , plus a value  $C$  that is *not* a linear function of the input stream but instead depends quadratically on  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$ . Moreover, the  $j$ th increment to  $C$  uses information that is available while processing the  $j$ th stream update, but not after the stream is gone. This is in contrast to linear sketches themselves, where the  $j$ th sketch update depends only on the  $j$ th stream update and no others.

**The Shaping Technique.** Another ubiquitous idea in streaming verification is the *shaping technique*, which transforms a data vector into a multidimensional array. This trick realizes  $g(X)$  as a summation of an even simpler multivariate polynomial: the latter can be evaluated directly by Verifier at several points, which forms the basis for his sketching. When applied to graph problems, this technique was historically used to reshape the  $\binom{n}{2}$ -dimensional vector of edge multiplicities. Recently, Chakrabarti and Ghosh [59] introduced the idea of reshaping the graph's *vertex space*, rather than just the edge space,

<sup>5</sup>This field is finite in the streaming verification literature, whereas traditional data streaming uses  $\mathbb{R}$ .

<sup>6</sup>Similiar nonlinearity was used recently in the more powerful model of 2-pass schemes [59].

thereby transforming the adjacency matrix into a 4-dimensional array. This trick was crucial to obtaining the first frugal schemes for TRIANGLECOUNT and MAXMATCHING.

**Our Contributions.** The new schemes in this work make the following contributions.

- We design new polynomial encodings for the graph-theoretic problems we study.
- We prominently employ nonlinear sketches, in the above sense, for almost all of our scheme designs.
- We use the shaping technique on the vertex space, often combining it with nonlinear sketching, thus expanding the applications of this very recent innovation.

Our solutions for TRIANGLECOUNT are particularly good illustrations of all of these ideas. Where Thaler’s nonlinear-sketch protocol treated each vertex as monolithic, our view of each vertex as an object in  $[t] \times [s]$  (for some pair  $t, s$  with  $t \cdot s = n$ ) let us do two things. In the laconic regime, we get to use Verifier’s increased space allowance in a way that Thaler’s protocol cannot, thereby extending his  $[n, n]$ -scheme to get an optimal tradeoff. In the frugal regime, it is significantly harder to exploit vertex-space shaping because Verifier cannot even afford to devote one entry per vertex in his linear sketches. We overcome this by finding a way for many vertices to “share” each entry of each linear sketch (see the string of equations culminating in eq. (4.18)), thus extending Thaler’s protocol to smoothly trade off communication for space.

We also extend the applicability of nonlinear sketching by identifying many further graph problems for which it yields significant improvements. Specifically, in Section 4.3.4, we describe two technical problems called INDUCEDGECOUNT and CROSSEGECOUNT, which are later used as primitives to optimally solve several important graph problems, including MAXMATCHING. We show how to apply sum-check with a nonlinear Verifier (see, e.g., eq. (4.27)) to optimally solve INDUCEDGECOUNT and CROSSEGECOUNT.

Finally, our schemes for SSSP feature a different kind of innovation on top of vertex-

space shaping and new, clever encodings of shortest-path problems in a manner amenable to sum-check. They overcome the frugal Verifier's space limitation by exploiting the Prover's room to generate a proof stream that mimics an iterative algorithm. For the Verifier to play along with such an iterative algorithm while lacking even one bit of space per vertex, a careful layering of fingerprint-based checks is needed on top of the sum-checks. We hope that our work here opens up possibilities for other instances of porting iterative algorithms to a streaming setting with the help of a prover.

### 4.3.2. Triangle Counting

**A frugal scheme.** We begin by describing a frugal scheme for TRIANGLECOUNT and then extend our ideas to obtain a sublinear scheme for the more general problem SUBGRAPH-COUNT. Throughout, we assume that the input is an  $n$ -vertex multigraph  $G = (V, E)$  with adjacency matrix  $A$ , built up through a stream of edge insertions and deletions.

Let  $T = T(G)$  be the number of triangles in  $G$  taking edge multiplicities into account, i.e., two triangles are considered distinct iff their corresponding sets of *edges* are distinct. Then,

$$6T = \sum_{v_1, v_2, v_3 \in V} A_{v_1 v_2} A_{v_2 v_3} A_{v_3 v_1}. \quad (4.13)$$

Let  $t$  and  $s$  be integer-valued parameters such that  $ts = n$ . Using a canonical bijection, we represent each vertex  $v \in V$  by a pair of integers  $(x, y) \in [t] \times [s]$ . This transforms the matrix  $A$  into a 4-dimensional array  $a$ , given by  $a(x_1, y_1, x_2, y_2) = A_{v_1 v_2}$ . Let  $\tilde{a}$  be the  $\mathbb{F}$ -extension of  $a$  for a sufficiently large finite field  $\mathbb{F}$  to be chosen later. Equation (4.13) now gives

$$6T = \sum_{x_1, x_2, x_3 \in [t]} p(x_1, x_2, x_3), \quad \text{where} \quad (4.14)$$

$$p(X_1, X_2, X_3) = \sum_{y_1, y_2, y_3 \in [s]} \tilde{a}(X_1, y_1, X_2, y_2) \tilde{a}(X_2, y_2, X_3, y_3) \tilde{a}(X_3, y_3, X_1, y_1). \quad (4.15)$$

Note that, for each  $i \in \{1, 2, 3\}$ , we have  $\deg_{X_i} p \leq 2t - 2$ . Thus, the number of monomials in  $p$  is at most  $(2t - 1)^3 \leq 8t^3$  and the total degree  $\deg p \leq 6t - 6 \leq 6t$ .

Our scheme for triangle counting operates as follows.

*Stream processing.* Verifier starts by picking  $r_1, r_2, r_3 \in_R \mathbb{F}$ . As the edge stream arrives, he maintains the three 2-dimensional arrays  $\tilde{a}(r_1, w, r_2, z)$ ,  $\tilde{a}(r_2, w, r_3, z)$ , and  $\tilde{a}(r_3, w, r_1, z)$ , for all  $(w, z) \in [s] \times [s]$  (using Fact 4.1.1). At the end of the stream, he uses these arrays to compute  $p(r_1, r_2, r_3)$ , using eq. (4.15).

*Help message.* Prover sends Verifier a polynomial  $\hat{p}(X_1, X_2, X_3)$  that she claims equals  $p(X_1, X_2, X_3)$ ; in particular, for each  $i \in \{1, 2, 3\}$ ,  $\deg_{X_i} \hat{p} \leq 2t - 2$ . She streams the coefficients of  $\hat{p}$  one at a time, according to some canonical ordering of the possible monomials.

*Verification and output.* As  $\hat{p}$  is streamed in, Verifier computes the check value  $C := \hat{p}(r_1, r_2, r_3)$  and the result value  $\hat{T} := \frac{1}{6} \sum_{x_1, x_2, x_3 \in [t]} \hat{p}(x_1, x_2, x_3)$ . If he finds that  $C \neq p(r_1, r_2, r_3)$ , he outputs  $\perp$ . Otherwise, he believes that  $\hat{p} \equiv p$  and accordingly, based on eq. (4.14), outputs  $\hat{T}$  as the answer.

The analysis of this scheme is along now-standard lines.

*Error probability.* Clearly, if Prover is honest (i.e.,  $\hat{p} \equiv p$ ), then the output is always correct. So the scheme errs only when  $\hat{p} \not\equiv p$  but Verifier's check passes. This means that the random point  $(r_1, r_2, r_3) \in \mathbb{F}^3$  is a root of the nonzero polynomial  $\hat{p} - p$ , which has total degree at most  $6t$ . By the Schwartz-Zippel Lemma (Fact 4.1.2), the probability of this event is at most  $6t/|\mathbb{F}| < 1/n$ , by choosing  $|\mathbb{F}|$  large enough.

*Help and Verification costs.* The number of bits used to describe the polynomial  $\hat{p}$  is the hcost. As noted, the polynomial  $\hat{p}$  has  $O(t^3)$  many coefficients, each of which is an element of  $\mathbb{F}$ , and hence has size  $O(\log n)$ . So the hcost is  $\tilde{O}(t^3)$ . The Verifier maintains three  $s \times s$  arrays, where each entry is an element of  $\mathbb{F}$ . Hence, the vcost

is  $\tilde{O}(s^2)$ . Therefore, we get a  $[t^3, s^2]$ -scheme for triangle counting, for parameters  $t, s$  with  $ts = n$ . Setting  $t = n^\alpha$  for  $\alpha \in (1/2, 2/3)$ , we get a  $(o(n^2), o(n))$ -scheme, which is frugal.

The result in this section is captured in the theorem below.

**Theorem 4.3.1.** *For any parameters  $t, s$  with  $ts = n$ , there is a  $[t^3, s^2]$ -scheme for TRIANGLECOUNT. In particular, there is an  $(o(n^2), o(n))$ -scheme for TRIANGLECOUNT.  $\square$*

This disproves Thaler's conjecture [164], which stated that TRIANGLECOUNT has no frugal scheme.

#### An improved frugal scheme.

**Theorem 4.3.2.** *There is an  $[nt^2, s]$ -scheme for TRIANGLECOUNT.*

Consider an adjacency matrix  $A$  of a graph on vertex set  $V$ . The addition of a new edge  $\{u, v\}$  creates  $\sum_{z \in V} A(u, z)A(v, z)$  new triangles.

Suppose that the input stream consists of  $L$  edge updates, the  $j$ th being  $(v_{1j}, v_{2j}, \Delta_j)$ ; recall that its effect is to add  $\Delta_j$  to the multiplicity of edge  $\{v_{1j}, v_{2j}\}$ . Suppose that the cumulative effect of the first  $j$  updates is to produce a multigraph  $G_j$  whose adjacency matrix is  $A_j$  and which has  $T_j$  triangles (counting multiplicity). As in Thaler's protocol [164], we can then account for the number of triangles added by the  $j$ th update:

$$T_j - T_{j-1} = \sum_{v_3 \in V} \Delta_j A_{j-1}(v_{1j}, v_3) A_{j-1}(v_{2j}, v_3).$$

As a result, the number of triangles  $T$  in the final graph  $G = G_L$  is

$$T = \sum_{j \in [L]} \sum_{v_3 \in V} \Delta_j A_{j-1}(v_{1j}, v_3) A_{j-1}(v_{2j}, v_3). \quad (4.16)$$

Our two new families of schemes for TRIANGLECOUNT apply the shaping technique to the above equation in two distinct ways, resulting in markedly different complexity behaviors.

We rewrite the variables  $v_{1j}$  and  $v_{2j}$  as pairs  $(x_{1j}, y_{1j})$  and  $(x_{2j}, y_{2j})$ , each in  $[t] \times [s]$  for parameters  $t, s$  with  $ts = n$ . The matrices  $A_{j-1}$  are now shaped into 3-dimensional arrays  $b_{j-1}$  that can be seen as functions on the domain  $[t] \times [s] \times [n]$ . As before, let  $\tilde{b}$  be an appropriate  $\mathbb{F}$ -extension. Working from eq. (4.16) and cleverly using the “unit impulse” function  $\delta$  seen in eq. (4.1),

$$\begin{aligned}
T &= \sum_{v_3 \in V} \sum_{j \in [L]} \Delta_j \tilde{b}_{j-1}(x_{1j}, y_{1j}, v_3) \tilde{b}_{j-1}(x_{2j}, y_{2j}, v_3) \\
&= \sum_{v_3 \in V} \sum_{w_1, w_2 \in [t]} \sum_{j \in [L]} \Delta_j \tilde{b}_{j-1}(w_1, y_{1j}, v_3) \tilde{b}_{j-1}(w_2, y_{2j}, v_3) \delta_{x_{1j}}(w_1) \delta_{x_{2j}}(w_2) \\
&= \sum_{v_3 \in V} \sum_{w_1, w_2 \in [t]} q(w_1, w_2, v_3), \quad \text{where}
\end{aligned} \tag{4.17}$$

$$q(W_1, W_2, V_3) = \sum_{j \in [L]} \Delta_j \tilde{b}_{j-1}(W_1, y_{1j}, V_3) \tilde{b}_{j-1}(W_2, y_{2j}, V_3) \delta_{x_{1j}}(W_1) \delta_{x_{2j}}(W_2). \tag{4.18}$$

We have a *multivariate* polynomial  $q(W_1, W_2, V_3)$ . We have the bounds  $\deg_{W_1} q \leq 2(t-1)$ ,  $\deg_{W_2} q \leq 2(t-1)$ , and  $\deg_{V_3} q \leq 2(n-1)$ , for a total degree of  $O(t+n) = O(n)$ . Importantly, the number of monomials in  $q$  is at most  $(2t-1)^2(2n-1) = O(nt^2)$ . We now present the corresponding scheme and its analysis.

*Stream processing.* Verifier picks  $r_1, r_2, r_3 \in_R \mathbb{F}$ . As the stream arrives, he maintains two 1-dimensional arrays:  $\tilde{b}_{j-1}(r_1, y, r_3)$  and  $\tilde{b}_{j-1}(r_2, y, r_3)$ , for all  $y \in [s]$  (using Fact 4.1.1). He also maintains an accumulator that starts at zero and, after the  $j$ th update  $(x_{1j}, y_{1j}, x_{2j}, y_{2j})$ , is incremented by

$$\Delta_j \tilde{b}_{j-1}(r_1, y_{1j}, r_3) \tilde{b}_{j-1}(r_2, y_{2j}, r_3) \delta_{x_{1j}}(r_1) \delta_{x_{2j}}(r_2).$$

By eq. (4.18), the final value of this accumulator is  $q(r_1, r_2, r_3)$ .

Notice that the accumulator is a nonlinear sketch of the input.

*Help message.* Prover sends Verifier a polynomial  $\hat{q}(W_1, W_2, V_3)$  that she claims equals  $q(W_1, W_2, V_3)$ . It should satisfy the degree bounds noted above. He lacks the space to store  $\hat{q}$ , so she streams the coefficients of  $\hat{q}$  in some canonical order.

*Verification and output.* As  $\hat{q}$  is streamed in, Verifier computes the check value  $C := \hat{q}(r_1, r_2, r_3)$  and the result value  $\hat{T} := \sum_{v_3 \in [n]} \sum_{w_1, w_2 \in [t]} \hat{q}(w_1, w_2, v_3)$ . If he finds that  $C \neq q(r_1, r_2, r_3)$ , he outputs  $\perp$ . Otherwise, he believes that  $\hat{q} \equiv q$  and accordingly, based on eq. (4.17), outputs  $\hat{T}$  as the answer.

*Error probability.* As before, we have perfect completeness and by the Schwartz–Zippel Lemma (Fact 4.1.2, this time using its full multivariate strength), this soundness error is at most  $\deg q / |\mathbb{F}| = O(n) / |\mathbb{F}| < 1/n$ , by choosing  $|\mathbb{F}|$  large enough.

*Help and Verification costs.* Prover can describe  $\hat{q}$  by listing its  $O(nt^2)$  coefficients. Verifier maintains two  $s$ -length arrays. Overall, we get an  $[nt^2, s]$ -scheme, as required.

### An Optimal Laconic Scheme.

**Theorem 4.3.3.** *There is a  $[t, ns]$ -scheme for TRIANGLECOUNT. This is optimal up to logarithmic factors.*

Let  $t, s \in \mathbb{N}$  be parameters with  $ts = n$ . We first consider rewriting the variable  $v_3$  in eq. (4.16) as a pair of integers  $(x_3, y_3) \in [t] \times [s]$  using some canonical bijection. This shapes each matrix  $A_{j-1}$  into a 3-dimensional array  $a_{j-1}$ , i.e., a function with domain  $[n] \times [t] \times [s]$ . Let  $\tilde{a}$  be the  $\mathbb{F}$ -extension of  $a$  for a sufficiently large finite field  $\mathbb{F}$  to be chosen

later. Then eq. (4.16) becomes

$$T = \sum_{j \in [L]} \sum_{x_3 \in [t]} \sum_{y_3 \in [s]} \Delta_j \tilde{a}_{j-1}(v_{1j}, x_3, y_3) \tilde{a}_{j-1}(v_{2j}, x_3, y_3) = \sum_{x_3 \in [t]} p(x_3), \quad \text{where} \quad (4.19)$$

$$p(X_3) = \sum_{j \in [L]} \sum_{y_3 \in [s]} \Delta_j \tilde{a}_{j-1}(v_{1j}, X_3, y_3) \tilde{a}_{j-1}(v_{2j}, X_3, y_3). \quad (4.20)$$

By the properties of  $\mathbb{F}$ -extensions observed above, we have the bound  $\deg p \leq 2(t-1)$ .

We now design our scheme as follows.

*Stream processing.* Verifier starts by picking  $r_3 \in_R \mathbb{F}$ . As the stream arrives, he maintains a 2-dimensional array of values  $\tilde{a}_{j-1}(v, r_3, y)$ , for all  $(v, y) \in [n] \times [s]$ , using Fact 4.1.1. He also maintains an accumulator that starts at zero and, after the  $j$ th update, is incremented by  $\Delta_j \sum_{y_3 \in [s]} \tilde{a}_{j-1}(v_{1j}, r_3, y_3) \tilde{a}_{j-1}(v_{2j}, r_3, y_3)$ . By eq. (4.20), the final value of this accumulator is  $p(r_3)$ .

*Help message.* Prover sends Verifier a polynomial  $\hat{p}(X_3)$  of degree  $\leq 2(t-1)$  that she claims equals  $p(X_3)$ .

*Verification and output.* Using Prover's message, Verifier computes the check value  $C := \hat{p}(r_3)$  and the result value  $\hat{T} := \sum_{x_3 \in [t]} \hat{p}(x_3)$ . If he finds that  $C \neq p(r_3)$ , he outputs  $\perp$ . Otherwise, he believes that  $\hat{p} \equiv p$  and accordingly, based on eq. (4.19), outputs  $\hat{T}$  as the answer.

The analysis of this scheme proceeds along standard lines long established in the literature.

*Error probability.* An honest Prover ( $\hat{p} \equiv p$ ) clearly ensures perfect completeness. The soundness error is the probability that Verifier's check passes despite  $\hat{p} \neq p$ , i.e., that the random point  $r_3 \in \mathbb{F}$  is a root of the nonzero degree- $(2t-2)$  polynomial  $\hat{p} - p$ . By

the Schwartz–Zippel Lemma (Fact 4.1.2), this probability is at most  $(2t - 2)/|\mathbb{F}| < 1/n$ , by choosing  $|\mathbb{F}|$  large enough.

*Help and Verification costs.* Prover describes  $\hat{p}$  by listing its  $O(t)$  many coefficients, spending  $O(t \log n)$  bits, since each is an element of  $\mathbb{F}$  and  $|\mathbb{F}| = n^{O(1)}$  suffices above. Verifier maintains an  $n \times s$  array whose entries are in  $\mathbb{F}$ , for a vcost of  $O(ns \log n)$ . Overall, we get a  $[t, ns]$ -scheme, as required.

### 4.3.3. Generalization: Counting Copies of an Arbitrary Subgraph

Now we consider the SUBGRAPHCOUNT $_k$  problem. Let  $H$  be a fixed  $k$ -vertex graph. The goal is to determine  $T_H = T_H(G)$ , the number of copies of  $H$  in the  $n$ -vertex multigraph  $G$  given by an input stream:  $n$  is growing whereas  $k = O(1)$ . As before, we take edge multiplicities into account.

Fix a numbering of the vertices of  $H$  as  $1, 2, \dots, k$ . Write  $i \sim j$  to denote  $\{i, j\} \in E(H) \wedge i < j$ . To generalize eq. (4.13), note that the expression  $\prod_{i \sim j} A_{v_i v_j}$  counts the number of copies of  $H$  occurring amongst vertices  $v_1, \dots, v_k$  in  $G$  where  $i \in V(H)$  is mapped to  $v_i \in V$ , provided that  $v_1, \dots, v_k$  are distinct. This subtlety of explicitly requiring the  $v_i$ s to be distinct did not arise for TRIANGLECOUNT because  $A_{v_1 v_2} A_{v_2 v_3} A_{v_3 v_1}$  is zero unless  $v_1, v_2, v_3$  are distinct. To enforce the distinctness condition in our more general setting, define an  $n \times n$  Boolean matrix  $B$  by  $B_{uv} = 1$  iff  $u \neq v$ . Then, defining  $\alpha_H$  to be the number of automorphisms of  $H$ ,

$$\alpha_H T_H = \sum_{v_1, \dots, v_k \in V} \left( \prod_{i \sim j} A_{v_i v_j} \right) \left( \prod_{i \neq j \in [k]} B_{v_i v_j} \right). \quad (4.21)$$

As before, we shape  $V$  into  $[t] \times [s]$  for parameters  $t$  and  $s$  with  $ts = n$ . This turns the 2-dimensional matrices  $A, B$  into 4-dimensional arrays  $a, b$ , which in turn have  $\mathbb{F}$ -extensions

$\tilde{a}, \tilde{b}$ . Equation (4.21) gives

$$\alpha_H T_H = \sum_{x_1, \dots, x_k \in [t]} p(x_1, \dots, x_k), \quad \text{where} \quad (4.22)$$

$$p(X_1, \dots, X_k) = \sum_{y_1, \dots, y_k \in [s]} \left( \prod_{i \sim j} \tilde{a}(X_i, y_i, X_j, y_j) \right) \left( \prod_{i \neq j \in [k]} \tilde{b}(X_i, y_i, X_j, y_j) \right). \quad (4.23)$$

For each  $i \in [k]$ ,  $\deg_{X_i} p \leq 2(k-1)(t-1) = O(t)$ . So the total degree  $\deg p = O(t)$  and  $p$  has at most  $O(t^k)$  monomials. This leads to a scheme for subgraph counting that naturally generalizes our earlier scheme for triangle counting. We sketch the salient features and the analysis.

*Stream processing.* Verifier picks  $r_1, \dots, r_k \in_R \mathbb{F}$  and maintains (using Fact 4.1.1)

$O(k^2) = O(1)$  many  $s \times s$  arrays:  $\tilde{a}(r_i, w, r_j, z)$  for each  $i \sim j \in [k]$  and  $\tilde{b}(r_i, w, r_j, z)$  for each  $i \neq j \in [k]$ , where  $(w, z) \in [s] \times [s]$ . The  $\tilde{b}$  arrays do not depend on the input stream and can be computed once and for all. At the end of the stream, he computes  $p(r_1, \dots, r_k)$  with the help of these values, using eq. (4.23).

*Help message.* Prover sends a polynomial  $\hat{p}(X_1, \dots, X_k)$  that she claims to be  $p(X_1, \dots, X_k)$ .

She streams the  $O(t^k)$  coefficients of  $\hat{p}$ , using some canonical ordering of the monomials.

*Verification and output.* Verifier computes the check value  $C := \hat{p}(r_1, \dots, r_k)$  and the

result value  $\hat{T}_H := \alpha_H^{-1} \sum_{x_1, \dots, x_k \in [t]} \hat{p}(x_1, \dots, x_k)$ . He outputs  $\perp$  if  $C \neq p(r_1, \dots, r_k)$ . Else, believing  $\hat{p} \equiv p$ , he outputs  $\hat{T}_H$  as the answer, in view of eq. (4.22).

*Error probability.* By a Schwartz-Zippel Lemma (Fact 4.1.2) argument as before, the error probability is at most  $\deg p / |\mathbb{F}| = O(t) / |\mathbb{F}| < 1/n$ , by choosing  $|\mathbb{F}|$  large enough.

*Help and Verification costs.* The hcost is  $\tilde{O}(t^k)$ , by the bound on the number of monomials in  $\hat{p}$ . Verifier stores  $O(1)$  many  $s \times s$  arrays, leading to a vcost of  $\tilde{O}(s^2)$ .

In summary, we obtain a  $[t^k, s^2]$ -scheme for counting copies of a fixed  $k$ -vertex subgraph  $H$ , for all choices of parameters  $t, s$  with  $ts = n$ . Setting  $t = n^{2/(k+2)}$  and  $s = n^{k/(k+2)}$  gives a scheme where both these costs are  $\tilde{O}(n^{2k/(k+2)})$ , which is  $o(n^2)$  for constant  $k$ . Thus, we get the following theorem.

**Theorem 4.3.4.** *For any parameters  $t, s$  such that  $ts = n$ , there is a  $[t^k, s^2]$ -scheme for  $\text{SUBGRAPHCOUNT}_k$ , where  $k$  is a constant. In particular, there is a sublinear scheme for  $\text{SUBGRAPHCOUNT}_k$  with total cost  $\tilde{O}(n^{2k/(k+2)})$ .  $\square$*

#### 4.3.4. A Technical Result: Counting Edges in Induced Subgraphs

We introduce two somewhat technical, though still natural, graph problems:  $\text{INDUCED-EDGECOUNT}$  and  $\text{CROSSEDGECOUNT}$ . We design schemes for these problems giving optimal tradeoffs (as usual, up to logarithmic factors). These schemes are key subroutines in our schemes for more standard, well-studied graph problems—such as  $\text{MAXMATCHING}$ —considered in Section 4.3.6.

The  $\text{INDUCED-EDGECOUNT}$  problem is defined as follows. The input is a stream of edges of a graph  $G = (V, E)$  followed by a stream of vertex subsets  $\langle U_1, \dots, U_\ell \rangle$  for some  $\ell \in \mathbb{N}$ , where  $U_i \subseteq V$  for  $i \in [\ell]$ . To be precise, the latter portion of the stream consists of the vertices of  $U_1$  in arbitrary order, followed by a delimiter, followed by the vertices of  $U_2$  in arbitrary order, and so on. The desired output is  $\sum_{i=1}^{\ell} |E(G[U_i])|$ , the sum of the numbers of edges in the induced subgraphs  $G[U_1], \dots, G[U_\ell]$ . Note that  $U_1, \dots, U_\ell$  need not be pairwise disjoint, so the sum may count some edges more than once.

The  $\text{CROSSEDGECOUNT}$  problem is an analog of the above for induced bipartite subgraphs. The input is a stream of edges followed by  $\ell$  pairs of vertex subsets  $\langle (U_1, W_1), \dots, (U_\ell, W_\ell) \rangle$ , where  $U_i \cap W_i = \emptyset$  for  $i \in [\ell]$ . The desired output is  $\sum_{i=1}^{\ell} |E(G[U_i, W_i])|$ , the sum of the number of cross-edges in the induced bipartite subgraphs  $G[U_1, W_1], \dots, G[U_\ell, W_\ell]$ . Note that the  $U_i$ s (or  $W_i$ s) need not be disjoint among themselves.

Importantly, in both of these problems, the edges *precede* the vertex subsets in the stream. This makes the problems intractable in the basic data streaming model. We shall prove the following results.

**Lemma 4.3.5.** *For any  $h, v$  with  $hv = n^2$ , there is an  $[h, v]$ -protocol for INDUCEDGE-COUNT.*

**Lemma 4.3.6.** *For any  $h, v$  with  $hv = n^2$ , there is an  $[h, v]$ -protocol for CROSSEGE-COUNT.*

**Scheme for INDUCEDGECOUNT (Proof of Lemma 4.3.5).** For the given instance, let  $M$  denote the desired output and let  $A$  be the adjacency matrix of  $G$ . For each  $i \in \ell$ , let  $B_i \in \{0, 1\}^V$  be the indicator vector of the set  $U_i$ , i.e.,  $B_i(v) = 1 \iff v \in U_i$ . Then,

$$M = \frac{1}{2} \sum_{i=1}^{\ell} \sum_{v_1, v_2 \in V} B_i(v_1) B_i(v_2) A(v_1, v_2). \quad (4.24)$$

Let  $t, s$  be integer parameters such that  $ts = n$ . We apply the shaping technique to eq. (4.24) by rewriting the variables  $v_j$  as pairs of integers  $(x_j, y_j) \in [t] \times [s]$ , for  $j \in \{1, 2\}$ . This transforms the matrix  $A$  into a 4-dimensional array  $a$  and each  $B_i$  into a 2-dimensional array  $b_i$ . Let  $\tilde{a}$  and  $\tilde{b}_i$  be the respective  $\mathbb{F}$ -extensions. Equation (4.24) now gives

$$2M = \sum_{i=1}^{\ell} \sum_{x_1, x_2 \in [t]} \sum_{y_1, y_2 \in [s]} \tilde{b}_i(x_1, y_1) \tilde{b}_i(x_2, y_2) \tilde{a}(x_1, y_1, x_2, y_2) = \sum_{x_1, x_2 \in [t]} p(x_1, x_2), \quad \text{where} \quad (4.25)$$

$$p(X_1, X_2) = \sum_{i=1}^{\ell} \sum_{y_1, y_2 \in [s]} \tilde{b}_i(X_1, y_1) \tilde{b}_i(X_2, y_2) \tilde{a}(X_1, y_1, X_2, y_2). \quad (4.26)$$

Our scheme exploits this expression in the same general manner as the analogous expressions for the TRIANGLECOUNT schemes from Section 4.3.2 (e.g., Equation (4.19)). Prover sends a bivariate polynomial  $\hat{p}(X_1, X_2)$ , which is claimed to be  $p$ , by streaming its

coefficients. Since  $\deg_{X_j} p \leq 2(t-1)$  for  $j \in \{1, 2\}$ , Prover need only send  $O(t^2)$  coefficients, for a help cost of  $\tilde{O}(t^2)$ . Verifier computes his output using eq. (4.25), giving perfect completeness. On the soundness side, Verifier checks the condition  $\hat{p}(r_1, r_2) = p(r_1, r_2)$  for randomly chosen  $r_1, r_2 \in_R \mathbb{F}$ . By the Schwartz-Zippel Lemma (Fact 4.1.2), the probability that he is fooled is at most  $\deg p/|\mathbb{F}| = O(t)/|\mathbb{F}| < 1/n$ , for the right choice of  $\mathbb{F}$ . It remains to describe how exactly Verifier evaluates  $p(r_1, r_2)$ , which we now address.

*Processing the stream of edges.* This is straightforward: Verifier maintains the 2-dimensional array of values  $\tilde{a}(r_1, w, r_2, z)$ , for all  $w, z \in [s]$ , using Fact 4.1.1.

*Processing the stream of vertex subsets.* Verifier initializes an accumulator to zero and allocates workspace for two arrays of length  $s$  with entries in  $\mathbb{F}$ . For each  $i \in [\ell]$ , as the vertices of  $U_i$  arrive, he maintains  $\tilde{b}_i(r_1, z)$  and  $\tilde{b}_i(r_2, z)$  for each  $z \in [s]$ , using that workspace. Upon seeing the delimiter marking the end of  $U_i$ , he computes

$$\sum_{y_1, y_2 \in [s]} \tilde{b}_i(r_1, y_1) \tilde{b}_i(r_2, y_2) \tilde{a}(r_1, y_1, r_2, y_2) \quad (4.27)$$

and adds this quantity to the accumulator. Note that the workspace is reused when the stream moves on from  $U_i$  to  $U_{i+1}$ . By eq. (4.26), after the last set  $U_\ell$  is streamed, the accumulator holds  $p(r_1, r_2)$ .

*Help and verification costs.* We argued above that the hcost is  $\tilde{O}(t^2)$ . Meanwhile, Verifier's storage is dominated by the  $s \times s$  array he maintains, leading to a vcost of  $\tilde{O}(s^2)$ .

Therefore, we obtain a  $[t^2, s^2]$ -scheme for any parameters  $t, s$  with  $ts = n$ . In other words, we get an  $[h, v]$ -scheme for any  $h, v$  with  $hv = n^2$ .

**Scheme for CROSSEGE COUNT (Proof of Lemma 4.3.6).** Our solution for INDUCED-EDGE COUNT can easily be modified to obtain a protocol for CROSSEGE COUNT with the

same costs. If  $B_i$  and  $C_i$  are the indicator vectors of the sets  $U_i$  and  $W_i$ , respectively, then the desired output is

$$M = \sum_{i=1}^{\ell} \sum_{v_1, v_2 \in V} B_i(v_1) C_i(v_2) A(v_1, v_2), \quad (4.28)$$

where we used the fact that each  $U_i \cap W_i = \emptyset$ . Since eq. (4.28) has essentially the same form as eq. (4.24), a scheme very similar to the previous one solves **CROSSEDGECOUNT**: Verifier simply keeps track of arrays corresponding to  $C_i$  alongside ones corresponding to  $B_i$ .

#### 4.3.5. Maximum Matching

We now turn to the **MAXMATCHING** problem, again giving a frugal scheme. Our input is an edge stream of an  $n$ -vertex graph  $G = (V, E)$  and we would like to determine  $\alpha'(G)$ , the cardinality of a maximum matching in  $G$ . We follow the broad outline of the semi-streaming scheme for **MAXMATCHING** by Thaler [164]. That scheme has two parts. In the first part, Prover convinces Verifier that  $\alpha'(G) \geq k$ , for some integer  $k$ . In the second part, she convinces him that  $\alpha'(G) \leq k$ . For the former, Prover simply provides a suitable matching  $M$  and convinces Verifier that  $M \subseteq E$  using the **SUBSET** scheme from Fact 4.1.3. For the latter, Prover uses the Tutte-Berge formula [49], which states that

$$\alpha'(G) = \frac{1}{2} \min_{U \subseteq V} \left( |U| + |V| - \text{odd}(G \setminus U) \right), \quad (4.29)$$

where  $\text{odd}(G \setminus U)$  denotes the number of connected components in  $G \setminus U$  with an odd number of vertices. The most challenging part of the scheme is evaluating  $\text{odd}(G \setminus U)$ , which involves the sub-problem of verifying whether all the connected components of a graph (as claimed by the Prover) are disconnected from each other. Thaler comments that this is the part that acts as a barrier in reducing the vcost to  $o(n)$  without increasing the

hcost to  $\Omega(n^2)$ . We present a novel frugal scheme for this sub-problem. The rest of the protocol solves the same sub-problems as the aforementioned paper. Most of their sub-schemes for these sub-problems, however, were trivial for  $\tilde{O}(n)$  space. We need schemes for the same problems that use only  $o(n)$  space and hence require more work. We describe our protocol below.

To convince the Verifier that the size of a maximum matching in  $G$  is  $k$ , Prover proves that it is (a) at least  $k$ , and (b) at most  $k$ . For (a), she simply sends (as a stream) a set  $M$  of  $k$  edges that constitutes a matching of  $G$ . Verifier can easily check using  $O(\log n)$  space that the set has size  $k$ . Next, he needs to check that  $M \subseteq E$ , and that  $M$  is indeed a matching. For the former, we can use the SUBSET scheme (Fact 4.1.3) and get an  $[h, v]$ -scheme, where  $v$  is the  $o(n)$  value we are aiming for and  $h = n^2/v$ . To verify that  $M$  is a matching, we check whether every vertex in  $M$  appears exactly once in this stream. Treating  $M$  as a stream of vertices, we can do this as follows: First, compute  $F_2$ , the second frequency moment of the stream, using an  $[h, v]$ -scheme where  $v$  is the  $o(n)$  vcost we want, and  $h = n/v$  ([57], Theorem 4.1). Next, verify that it equals  $2k$  (this happens iff all  $2k$  elements are distinct).

For (b), we apply eq. (4.29). Prover sends  $U^* \subseteq V$  and claims that  $k = \frac{1}{2}(|U^*| + |V| - \text{odd}(G \setminus U^*))$ . To check this, Verifier just needs to compute  $\text{odd}(G \setminus U^*)$ . We do this in the following way.

Let  $[C]$  be the set of  $C$  connected components of  $G \setminus U^*$ . For  $c \in [C]$  and  $u \in G \setminus U^*$ , Prover sends an array  $L$  of pairs  $(c, u)$  such that  $u \in c$ . The array  $L$  is sorted in non-decreasing order of  $c$ , i.e., she first sends the vertices in connected component 1, followed by those in component 2, and so on. If  $L$  is indeed as Prover claims, then  $\text{odd}(G \setminus U^*)$  is equal to the number of components  $c$  that arrive with an odd number of vertices in  $L$ . Since  $L$  is sorted with respect to  $c$ , Verifier can count this number easily using  $O(\log n)$  space. He can verify that the vertices in the tuples of  $L$  constitute  $G \setminus U^*$ , and that no vertex  $u$  is

repeated in different tuples of  $L$ , using frugal schemes implied by the standard protocols mentioned above.

Thus, it only remains to verify that  $L$  is as claimed. For this, we need to check whether the following two properties hold:

- (i) For each  $c \in [C]$ , the vertices in  $G \setminus U^*$  that are claimed to be in component  $c$  are all connected in  $G \setminus U^*$ .
- (ii) For every pair  $(u, v)$  of vertices in  $G \setminus U^*$  that are claimed to be in different components, we have  $(u, v) \notin E$ .

For Property (i), Prover sends a spanning tree for each connected component  $c$  and Verifier can check if all of them are valid using an  $[n^{1+\alpha}, n^{1-\alpha}]$ -scheme, for any  $\alpha \in [0, 1]$  ([57], Theorem 7.7) so as to get the desired  $o(n)$  vcost.

Checking Property (ii) is the most challenging part. We give a novel protocol for this part that uses  $o(n)$  vcost and  $o(n^2)$  hcost. Slightly abusing notation, consider the array  $L$  in the form of a  $C \times |G \setminus U^*|$  matrix, such that  $L_{cu} = 1$  if  $u \in c$ , and  $L_{cu} = 0$  otherwise. Denote the ones' complement of this matrix by  $\bar{L}$ . Let  $A$  be the adjacency matrix of  $G \setminus U^*$ . Finally, let  $\gamma$  denote the total number of cross edges that go between two connected components in  $G \setminus U^*$ . Then, we have

$$2\gamma = \sum_{\substack{c \in [C] \\ u, v \in G \setminus U^*}} L_{cu} \bar{L}_{cv} A_{uv}. \quad (4.30)$$

Property (ii) is satisfied iff  $\gamma = 0$ . Recalling that  $C = O(n)$  and  $|G \setminus U^*| = O(n)$ , we note that eq. (4.30) has a similar form as that of eq. (4.13). Thus, it can be exploited in essentially the same way as the  $[t^3, s^2]$ -scheme for TRIANGLECOUNT, for parameters  $t, s$  with  $ts = n$ . Once again, setting  $t = n^\alpha$  for  $\alpha \in (1/2, 2/3)$ , we get a frugal scheme.

The next theorem summarizes the result in this section.

**Theorem 4.3.7.** *For any parameters  $t, s$  with  $ts = n$ , there is a  $[t^3, s^2]$ -scheme for MAX-MATCHING. In particular, there is an  $(o(n^2), o(n))$ -scheme for MAXMATCHING.  $\square$*

This disproves yet another conjecture of Thaler [164], which stated that MAXMATCHING has no frugal scheme.

**Optimal Frugal Scheme.** To optimally check that the purported connected components of  $H$  are indeed disconnected from each other, we use the INDUCEDGECOUNT scheme as a subroutine. Prover streams the vertices in  $H$  by listing its connected components in some order  $\langle U_1, \dots, U_\ell \rangle$ . Verifier uses Lemma 4.3.5 to count  $m_1 := |E(H)|$  (invoking that lemma with a single subset  $V(H)$ ). In parallel, using the same scheme, Verifier computes the sum  $m_2 = \sum_{i=1}^\ell |E(G[U_i])|$ . The subsets  $U_i$  are pairwise disconnected iff  $m_2 = m_1$ , which Verifier checks. The sub-checks of whether  $U_i$ s are indeed pairwise disjoint (as sets) and whether  $U^* \sqcup V(H) = V(G)$  can be done via fingerprinting (as in section 4.1).

*Help and verification costs.* Prover streams  $U^*$  and the vertices in  $H$  in a certain order, which adds  $O(n \log n)$  bits to the hcost of the INDUCEDGECOUNT protocol. The vcost stays the same, asymptotically, giving us an  $[n + h, v]$ -scheme for MAXMATCHING for any  $h, v$  with  $hv = n^2$ . Overall, we have established the following theorem.

**Theorem 4.3.8.** *There is an  $[nt, s]$ -scheme for MAXMATCHING. This is optimal up to logarithmic factors, since any  $(h, v)$ -scheme is known to require  $hv = \Omega(n^2)$  [57].*

**Protocol for Space Larger Than  $n$ .** There is no laconic scheme known for the general MAXMATCHING problem. The barrier seems to be that a natural witness for the problem is an actual maximum matching of the graph, which can be of size  $\Theta(n)$ . We show that large maximum matching size  $\alpha'(G)$  is indeed the sole barrier to obtaining a laconic scheme. In particular, for any graph  $G$ , we give a scheme for MAXMATCHING with hcost  $\alpha'(G)$ . This yields a laconic scheme for the case when  $\alpha'(G) = o(n)$ .

Let  $H = G \setminus U^*$  as above, and let  $U_1, \dots, U_\ell$  be the connected components of  $H$ .

By the Tutte-Berge formula (eq. (4.29)), we have  $2k = |U^*| + (n - \text{odd}(H))$ . This leads to the following observations.

**Observation 4.3.9.**  $|U^*| = O(k)$ .

**Observation 4.3.10.** *The number of edges in a spanning forest of  $H$  is  $|V(H)| - \ell \leq n - \text{odd}(H) = O(k)$ .*

We now describe our protocol, which is along the lines of the protocol above, but this time we crucially use the fact that we are allowing Verifier a space usage of  $v \geq n$ .

To show that  $\alpha'(G) \geq k$ , Prover sends a matching  $M$  of size  $k$ . Verifier stores  $M$  explicitly and checks that it is indeed a matching. Then, he verifies that  $M \subseteq E$  using the Subset Scheme (Fact 4.1.3). Therefore, this part of the scheme uses  $\text{hcost } \tilde{O}(k + h)$  and  $\text{vcost } \tilde{O}(v)$  for any  $h, v$  with  $hv = n^2$  and  $v \geq n$ .

Recall that to show that  $\alpha'(G) \leq k$ , it suffices to compute  $\text{odd}(H)$ . Prover sends the set  $U^*$ . By Observation 4.3.9, this takes  $\tilde{O}(k)$   $\text{hcost}$ . Verifier has  $\Omega(n)$  space, and hence, he can store  $V \setminus U^* = V(H)$ . Next, Prover sends a spanning forest  $F$  of  $H$ . By Observation 4.3.10, this again incurs  $\text{hcost } \tilde{O}(k)$ . Verifier stores  $F$  and verifies that  $F \subseteq E$  using the Subset Scheme (Fact 4.1.3). From  $F$ , Verifier explicitly knows the purported connected components  $U_1, \dots, U_\ell$  of  $H$ . He finally verifies that  $U_i$ 's are disconnected from each other by checking that all edges in  $H$  are contained in these components. He can do this by checking whether  $|E \cap (V(H) \times V(H))| = |E \cap (\cup_{i=1}^\ell U_i \times U_i)|$  using the Intersection Scheme (Fact 4.1.3). If the check passes he goes over the  $U_i$ s to compute  $\text{odd}(H)$  and thus, this part can also be solved using a  $[k + h, v]$  scheme for any  $h, v$  with  $hv = n^2$  and  $v \geq n$ . Hence, we obtain the following theorem.

**Theorem 4.3.11.** *For any  $h, v$  with  $hv \geq n^2$  and  $v \geq n$ , there is an  $[\alpha' + h, v]$ -scheme for MAXMATCHING, where  $\alpha'$  is the size of the maximum matching of the input graph. In particular, there is an  $[\alpha', n^2/\alpha']$ -scheme.*

### 4.3.6. Applications to Other Graph Problems

In Section 4.3.5, we used a scheme for INDUCEDGECOUNT to obtain an optimal frugal scheme for MAXMATCHING. Below, we give applications of edge-counting schemes to several other well-studied graph problems.

**Triangle-Counting.** A scheme for TRIANGLECOUNT follows immediately from INDUCEDGECOUNT. For  $v \in [n]$ , set the subsets  $U_v = N(v)$ , the neighborhood of vertex  $v$ . Then, observe that INDUCEDGECOUNT returns three times the total number of triangles in the graph. The sets  $U_v$ , however, need to be sent in some order by Prover, and so the additional hcost to INDUCEDGECOUNT is  $\tilde{O}(\sum_v |N(v)|) = \tilde{O}(m)$ . As Prover basically repeats the edge stream in a different order, we can check if it's consistent with the input stream by fingerprinting (see Section 4.1). Hence, we get an  $[m + h, v]$ -scheme for any  $h, v$  with  $hv = n^2$ .

**Theorem 4.3.12.** *For any  $h, v$  with  $hv \geq n^2$ , there is an  $[m + h, v]$ -scheme for TRIANGLECOUNT. In particular, there is an  $[m, n^2/m]$ -scheme.*

The only other scheme for TRIANGLECOUNT achieving  $hv = n^2$  tradeoff with  $\text{vcost} = o(n)$  was an  $[n^2, 1]$ -scheme by Chakrabarti et al. [57]. Our result generalizes it for any graph with  $m$  edges, thus achieving a better hcost and a smooth tradeoff for sparse graphs.

We note that in the above scheme, Prover needs to send the sets  $U_v = N(v)$  because the INDUCEDGECOUNT protocol needs the neighborhood of each vertex to arrive contiguously in the stream. This is essentially the input stream order in the *adjacency-list* or the *vertex-arrival* streaming model. Thus, for the problem TRIANGLECOUNT-ADJ, Verifier gets the  $U_v$ s in the desired order as part of the input; so Prover need not repeat them, saving the huge  $\tilde{O}(m)$  hcost. However, there is another issue in directly applying the INDUCEDGECOUNT subroutine in this case. In the definition of INDUCEDGECOUNT, we assume that all the edges in the graph arrive before the vertex subsets  $U_i$ . Here, the  $U_v$ s and the edges arrive in interleaved manner (although each  $U_v$  arrives contiguously). But

we show that we can still apply the scheme for INDUCEDGECOUNT to get the desired output. Let the order in which the  $U_v$ s appear be  $\langle U_1, \dots, U_n \rangle$ , and let  $G_v$  denote the graph consisting of edges seen till the arrival of  $U_v = N(v)$ . Then, applying INDUCEDGECOUNT, what we count is

$$\sum_{v \in [n]} |E(G_v[N(v)])| = \sum_{v \in [n]} \#\{\text{triangles incident on } v \text{ in } G_v\} = 2T.$$

The last equality follows since every triangle whose vertices appear in the order  $\langle v_1, v_2, v_3 \rangle$  will be counted twice: once when  $v_2$  arrives and once when  $v_3$  arrives. We therefore obtain the following theorem.

**Theorem 4.3.13.** *For any  $h, v$  with  $hv \geq n^2$ , there is an  $[h, v]$ -scheme for TRIANGLECOUNT-ADJ.*

**Maximal Independent Set (MIS).** Recent works [17, 70] have studied the problem of finding a maximal independent set in the basic data streaming model. They show a lower bound of  $\Omega(n^2)$  for a one-pass streaming algorithm. This implies a lower bound of  $hv \geq n^2$  for any  $[h, v]$ -scheme for MIS. Hence, we aim for  $hv = n^2$  and describe a frugal scheme using INDUCEDGECOUNT. Since the output size of the problem can be  $\Theta(n)$ , it would only make sense in the frugal regime if the Prover sends the output as a stream and the Verifier checks that it is valid using  $o(n)$  space.

Let  $U$  be an MIS in the graph  $G$ . Prover sends  $U$  and Verifier uses INDUCEDGECOUNT to count the number of edges in  $G[U]$  and verifies that it equals 0. If the check passes,  $U$  is indeed an independent set. It remains to check the maximality of  $U$ . If  $U$  is maximal, then, for each vertex  $v$  in  $G \setminus U$ , there must be a vertex  $u$  in  $U$ , such that  $(v, u)$  is an edge. Prover points out such a vertex  $u \in U$  for each  $v \in G \setminus U$ . Let  $F$  denote this set of  $|G \setminus U|$  purported edges. Now, we use Subset Scheme (Fact 4.1.3) to verify that  $F \subseteq E$ , i.e., all these edges are actually present in  $G$ . We can use fingerprinting (as in

Section 4.1) to check that  $F$  contains an edge for each vertex in  $G \setminus U$  and the Intersection Scheme to verify that the set of their partners is disjoint from  $G \setminus U$ , i.e., belong to  $U$ . Thus, the additional hcost to INDUCEDEDGECOUNT, Subset, and Intersection Schemes is  $\tilde{O}(n)$ , the number of bits required to send  $U$  and  $F$ . Therefore, by Lemma 4.3.5, we get an  $[n + h, v]$ -scheme for MIS for any  $h, v$  with  $hv = n^2$ . Thus, our scheme is optimal for the frugal regime.

**Theorem 4.3.14.** *For any  $t, s$  with  $ts = n$ , there is an  $[nt, s]$ -scheme for MIS. This is optimal up to logarithmic factors, since any  $(h, v)$ -scheme is known to require  $hv = \Omega(n^2)$ .*

**Acyclicity Testing and Topological Sorting.** We now turn to the ACYCLICITY problem in directed graphs. It is easy to prove that a graph is *not* acyclic by showing the existence of a cycle  $C$ . Verifier checks that  $C \subseteq E$  using Subset Scheme (Fact 4.1.3). Hence, this can be done using an  $[h, v]$ -scheme for any  $h \geq |C|$ .

The more interesting case is when the graph is indeed acyclic. Note that a directed graph is acyclic if and only if it has a topological ordering. Thus, it suffices to show a valid topological ordering of the vertices. TOPOSORT is a fundamental graph algorithmic problem of independent interest. ACYCLICITY has a one-pass lower bound of  $\Omega(n^2)$  in the basic data streaming model. Recently, Chakrabarti et al. [60] showed that TOPOSORT also requires  $\Omega(n^2)$  space in one pass. These translate to a lower bound of  $hv \geq n^2$  for any  $[h, v]$ -scheme for these problems. Hence, we aim for a scheme with  $hv = n^2$  and design a protocol for TOPOSORT in the frugal regime. Since this problem has output size  $\tilde{\Theta}(n)$ , we aim for a protocol where Prover sends a topological ordering of the graph and Verifier checks its validity using  $o(n)$  space. Moreover, this protocol can be used for the YES case of ACYCLICITY.

Verifier uses CROSSEGECOUNT to solve this. As Prover sends the topological order  $\langle v_1, \dots, v_n \rangle$ , for each  $i \in [n - 1]$ , Verifier sets  $U_i = \{v_1, \dots, v_i\}$  and  $W_i = \{v_{i+1}\}$  for CROSSEGECOUNT. Thus, the protocol counts precisely the number of forward edges

induced by the ordering. If it equals  $m$ , then the ordering is indeed a valid topological order. Note that since  $U_{i+1} = U_i \cup \{v_{i+1}\}$ , Prover doesn't need to send  $U_{i+1}$  afresh; just  $v_{i+1}$  is enough for Verifier to update his sketch. Verifier can use fingerprinting (see Section 4.1) to make sure that precisely the set  $V$  was sent in some order. Hence, the additional hcost to **CROSSEGECOUNT** is the number of bits required to express the topological order, i.e.,  $\tilde{O}(n)$ . Therefore, by Lemma 4.3.6, we get a  $[n + h, v]$ -scheme for any  $hv = n^2$ .

**Theorem 4.3.15.** *For any  $t, s$  with  $ts = n$ , there is an  $[nt, s]$ -scheme for **TOPOSORT**. This is optimal up to logarithmic factors, since any  $(h, v)$ -scheme is known to require  $hv = \Omega(n^2)$ .*

**Corollary 4.3.16.** *For any  $t, s$  with  $ts = n$ , there is an  $[nt, s]$ -scheme for **ACYCLICITY**. This is optimal up to logarithmic factors, since any  $(h, v)$ -scheme is known to require  $hv = \Omega(n^2)$ .*

For dense graphs, our result generalizes the  $[m, 1]$ -scheme of Cormode et al. [71] for **ACYCLICITY** by achieving a smooth tradeoff.

**Graph Connectivity.** The graph connectivity problem has garnered considerable attention in the basic and annotated streaming settings [5, 57, 164]. For any  $t, s$  with  $ts = n$ , Chakrabarti et al. [57] gave an  $[nt, s]$ -scheme that determines whether an input graph is connected or not. Their scheme cannot, however, solve the more general problem of returning the number of connected components. The  $[t^3, s^2]$ -scheme (for any  $ts = n$ ) of Chakrabarti and Ghosh [59] does solve this problem, but has a worse tradeoff. As noted in Section 4.3.5, we can use **INDUCEDEDGECOUNT** to check that all purported connected components are indeed disconnected from each other. On the other hand, the scheme of Chakrabarti et al. [57] can check whether each component is actually connected. Hence, we can verify the number of connected components claimed by Prover by running these schemes parallelly. Thus, we generalize the result of Chakrabarti et al. [57] by obtaining an  $[nt, s]$ -scheme for counting the number of connected components of a graph.

**Theorem 4.3.17.** *For any  $t, s$  with  $ts = n$ , there is an  $[nt, s]$ -scheme for counting the number of connected components of a graph.*

#### 4.3.7. Path Problems

In this section, we focus on path-related problems. Specifically, we study ST-KPATH for  $k \geq 3$  and the fundamental ST-SHORTESTPATH problem. Simple reductions from the INDEX<sub>N</sub> problem, for  $N = n^2$ , show that a one-pass algorithm for either of these problems would require  $\Omega(n^2)$  space in the basic (sans prover) streaming model. They also show that a one-pass scheme would require a total cost of  $\Omega(n)$ . We present a scheme for ST-KPATH for general  $k$  that can also be used to solve ST-SHORTESTPATH. It is a semi-streaming scheme when  $k$  is polylogarithmic in  $n$ , and hence matches the lower bound (up to polylogarithmic factors). Next, we explore if we can break the  $\Omega(n)$  barrier for schemes for ST-KPATH at the cost of allowing a few more passes over the input. We achieve this for constant  $k$  by generalizing the protocol for ST-3PATH. We present all our schemes for undirected graphs, but they can easily be modified to work for directed graphs as well.

**A Semi-Streaming Scheme for Detecting Short Paths.** For ST-3PATH, it is easy to obtain a semi-streaming scheme by checking (using Fact 4.1.3) whether the set  $N[v_s] \times N[v_t]$  and the edge set  $E$  are disjoint. For  $k > 3$ , things are not that direct and we require more work. We describe the protocol below for a multigraph  $G$ .

Let  $A$  denote the adjacency matrix of the multigraph  $G$  and let  $\tilde{A}$  be the  $\mathbb{F}$ -extension of  $A$ , for some large finite field  $\mathbb{F}$ . For  $u \in N_{i+1}(v_s)$ , let  $d_{u,i}$  be the number of (in-)neighbors of  $u$  in  $N_i(v_s)$ . It follows that

$$d_{u,i} = \sum_{v \in N_i(v_s)} A(v, u). \quad (4.31)$$

We are now ready to describe the protocol.

*Stream processing.* Verifier picks  $r \in_R \mathbb{F}$  and stores  $\tilde{A}(v, r)$  for each  $v \in [n]$ , maintaining them dynamically as the stream arrives (using Fact 4.1.1). He also stores the set  $N_1(v_s)$ .

*Help message.* At the end of the stream, Prover sends Verifier  $k-1$  polynomials  $\hat{p}_1, \dots, \hat{p}_{k-1}$ , and she claims  $\hat{p}_i \equiv p_i$  for each  $i \in [k]$ , where

$$p_i(U) = \sum_{v \in N_i(v_s)} \tilde{A}(v, U). \quad (4.32)$$

*Verifier's computation.* Verifier iteratively constructs  $N_i(v_s)$  for  $i \in [k]$ . Each time, after computing  $N_i(v_s)$  for a distance parameter  $i$ , he checks whether  $v_t \in N_i(v_s)$ . If so, he stops and outputs YES. Otherwise, he proceeds to compute  $N_{i+1}(v_s)$ . If he finds that  $\forall i \in [k] : v_t \notin N_i(v_s)$ , then he outputs NO. The inductive neighborhood computation is done as follows.

Assume that Verifier has the set  $N_i(v_s)$  for some  $i \in [k-1]$ ; this holds initially, since he has stored  $N_1(v_s)$ . He computes  $p_i(r)$  using Equation (4.32) and checks whether  $\hat{p}_i(r) = p_i(r)$ . If the check passes, he believes that  $\hat{p}_i \equiv p_i$  and evaluates  $\hat{p}_i(u)$  for each  $u \in V$ . By eq. (4.31),  $p_i(u)$  equals  $d_{u,i}$ , which is non-zero iff  $u \in N_{i+1}(v_s)$ . Hence, he sets  $N_{i+1}(v_s) = \{u : \hat{p}_i(u) \neq 0\}$ .

*Error probability.* The protocol errs when we have  $\hat{p}_i \not\equiv p_i$  for some  $i$ , but Verifier's check passes. This implies that  $r$  is a root of the non-zero polynomial  $\hat{p}_i - p_i$ . For a given  $i$ , the total degree of this polynomial is at most  $2n$ . Then, probability that  $r$  is a root is at most  $2n/|\mathbb{F}| < 1/n^2$ , for large enough choice of  $|\mathbb{F}|$ . Taking a union bound over all  $i \in [k]$ , we get that the probability that  $r$  is a root of  $\hat{p}_i - p_i$  for some  $i$  is at most  $1/n$ .

*Help and Verification costs.* Since the degree of each  $p_i$  is  $\leq 2n$ , the total hcost is  $\tilde{O}(kn)$ . Verifier stores  $\tilde{A}(v, r)$  for each  $v \in [n]$ , which requires  $\tilde{O}(n)$  space. Ad-

ditionally, to compute  $N_{i+1}(v_s)$  for some  $i \in [k]$ , he needs only the set  $N_i(v_s)$ . Thus, we can store the  $N_i(v_s)$  sets by reusing space repeatedly, and this requires  $O(n)$  space. Hence, the total vcost of this protocol is  $\tilde{O}(n)$ . Therefore, we get a  $[kn, n]$ -scheme for checking for the existence of a path of length at most  $k$  from  $v_s$  to  $v_t$ .

**Theorem 4.3.18.** *Given an  $n$ -vertex (directed or undirected) multigraph  $G(V, E)$  and specified vertices  $v_s, v_t \in V$ , for any  $k \leq n - 1$ , there is a  $[kn, n]$ -scheme for ST-KPATH. In particular, there is a semi-streaming scheme for ST-KPATH when  $k$  is polylogarithmic in  $n$ .  $\square$*

**Applications.** Based on the scheme in Theorem 4.3.18, we have the following straightforward corollaries. Contrast these results with Theorem 7 of Cormode et al. [71]. They give an  $[h, v]$ -scheme for a *weighted* version of ST-SHORTESTPATH for any  $h, v$  such that  $hv \geq Dn^2$  and  $h \geq Dn$ , where  $D$  is the maximum distance from  $v_s$  to any other vertex reachable from it. A similar result holds for  $v_s$ - $v_t$  connectivity in directed graphs with diameter  $D$ . Their schemes work only for simple graphs, whereas ours naturally work for multigraphs; on the other hand, we only solve the unweighted version of the problem. Notably, there is a significant difference in the underlying techniques: their schemes are based on linear programming duality, while we have a more directly algebraic approach.

**Corollary 4.3.19.** *Given a (directed or undirected) multigraph  $G(V, E)$ , with edge multiplicities polylogarithmic in  $n$ , and specified vertices  $v_s, v_t \in V$ , there is a  $[kn, n]$ -scheme for ST-SHORTESTPATH, where  $k$  is length of the shortest  $v_s$ - $v_t$  path.*

*Proof.* If there is no  $v_s$ - $v_t$  path, Prover sends the connected component  $C$  that  $v_s$  is in. Verifier first checks that  $C$  is indeed connected ([57], Theorem 7.7). Next, he verifies that there is no edge going out from  $C$  by checking whether the set  $C \times (V \setminus C)$  and the edge set  $E$  are disjoint (Fact 4.1.3). Both of these are  $[n, n]$ -schemes.

If there is a  $v_s$ - $v_t$  path, and the shortest such path  $H$  has length  $k$ , then Prover sends it to Verifier, who checks whether  $H$  is indeed a  $v_s$ - $v_t$  path and whether  $H \subseteq E$  using an

$[n, n]$ -scheme, using the polylogarithmic bound on the edge multiplicities (Fact 4.1.3). In parallel, he uses a  $[kn, n]$ -scheme to verify that there is no  $v_s$ – $v_t$  path of length at most  $k - 1$  (Theorem 4.3.18).  $\square$

**Corollary 4.3.20.** *Given a directed  $n$ -vertex multigraph  $G$ , with edge multiplicities polylogarithmic in  $n$ , there is a  $[Dn, n]$ -scheme for checking  $v_s$ – $v_t$  connectivity, where  $D$  is the maximum distance from  $v_s$  to any other vertex reachable from it. In particular, there is a semi-streaming scheme for checking  $v_s$ – $v_t$  connectivity in a directed multigraph with diameter polylogarithmic in  $n$ .*

*Proof.* If there is a  $v_s$ – $v_t$  path  $H$ , then Prover sends it to the Verifier, and he can check whether  $H \subseteq E$  using an  $[n, n]$ -scheme, as edge multiplicity is polylogarithmic in  $n$  (Fact 4.1.3). If not, then we verify that there is no  $v_s$ – $v_t$  path of length at most  $D$  using a  $[Dn, n]$ -scheme (Theorem 4.3.18).  $\square$

**Unweighted Shortest Path.** We shall design a scheme that works even if the same edge appears multiple times in the stream (unlike prior work [71] that assumes that an edge appears at most once).

Prover sends distance labels  $\widehat{\text{dist}}[v]$  for all  $v \in V$ , claiming that  $\widehat{\text{dist}}[v] = \text{dist}(v_s, v)$ , the actual distance from the source vertex  $v_s$  to  $v$ . Let the radius- $d$  ball around  $v_s$  be  $B_d := \{v \in V : \text{dist}(v_s, v) \leq d\}$  and let  $\mathcal{B} := \{B_d : d \in [D]\}$  be the family of such balls. Let  $\hat{B}_d$  be the corresponding balls implied by Prover's  $\widehat{\text{dist}}$  labels, and  $\hat{\mathcal{B}} := \{\hat{B}_d : d \in [D]\}$ .

To check correctness, Verifier uses fingerprinting (Section 4.1) modified as follows. Letting  $B, \hat{B}$  also denote the respective characteristic vectors, define fingerprint polynomials

$$\varphi_B(X, Y) := \sum_{i \in [n]} \sum_{d \in [D]} B_d(i) X^i Y^d, \quad \varphi_{\hat{B}}(X, Y) := \sum_{i \in [n]} \sum_{d \in [D]} \hat{B}_d(i) X^i Y^d,$$

As the  $\widehat{\text{dist}}$  labels are streamed, Verifier constructs the fingerprint  $\varphi_{\hat{B}}(\beta_1, \beta_2)$  for some  $\beta_1, \beta_2 \in_R \mathbb{F}$ .

Over the course of the protocol, using further help from Prover, Verifier will construct the sets  $B_d$  inductively and, in turn, the “actual” fingerprint  $\varphi_B(\beta_1, \beta_2)$ . The next claim shows that comparing this with  $\varphi_{\hat{B}}(\beta_1, \beta_2)$  validates Prover’s  $\widehat{\text{dist}}$  labels.

**Claim 4.3.21.** *If  $\hat{B}_d = B_d$  for all  $d$ , then  $\widehat{\text{dist}}[v] = \text{dist}(v_s, v)$  for all vertices  $v$ .*

*Proof.* Suppose not. Let  $d^*$  be the smallest  $d$  such that  $\exists u \in B_{d^*}$  with  $\widehat{\text{dist}}[u] \neq \text{dist}(v_s, u)$ . Therefore,  $\text{dist}(v_s, u) = d^*$ . Now,  $d^*$  cannot be 0 since  $v_s$  is the only vertex in  $B_0$  and Verifier would reject immediately if  $\widehat{\text{dist}}(v_s) \neq 0$ . Since  $B_{d^*} = \hat{B}_{d^*}$ , we have  $u \in \hat{B}_{d^*}$ . This means  $\widehat{\text{dist}}(u) \leq d^*$ . Since  $\widehat{\text{dist}}(u) \neq d^*$ , we have  $\widehat{\text{dist}}(u) \leq d^* - 1$ . Thus,  $u \in \hat{B}_{d^*-1}$ , i.e.,  $u \in B_{d^*-1}$ , which is a contradiction to the minimality of  $d^*$ .  $\square$

As before,  $A$  denotes the adjacency matrix of the graph. Putting

$$q_d(u) := \sum_{v \in V} B_d(v) A(v, u), \text{ for each } u \in V, \quad (4.33)$$

$$\text{we have } B_{d+1} = \{u \in V : q_d(u) \neq 0\}. \quad (4.34)$$

To apply the shaping technique to (4.33), rewrite  $v$  as  $(x, y) \in [t] \times [s]$ . This reshapes  $A$  into a  $t \times s \times n$  array  $a(x, y, u)$  and  $B_d$  into a  $t \times s$  array  $b_d(x, y)$ . As usual, let  $\tilde{a}$  and  $\tilde{b}_d$  be the respective  $\mathbb{F}$ -extensions for a suitable finite field  $\mathbb{F}$ . Then, eq. (4.33) gives

$$q_d(u) = \sum_{x \in [t]} p_d(x, u), \quad \text{where} \quad (4.35)$$

$$p_d(X, U) := \sum_{y \in [s]} \tilde{b}_d(X, y) \tilde{a}(X, y, U). \quad (4.36)$$

*Stream processing.* Verifier picks  $r_1, r_2 \in_R \mathbb{F}$  and maintains  $\tilde{a}(r_1, y, r_2)$ . When he sees vertices in  $B_1$ , i.e.,  $v_s$  and its neighbors, he maintains  $b_1(r_1, y)$  for all  $y \in [s]$  and also updates the fingerprint  $\varphi_B(\beta_1, \beta_2)$  accordingly.

Verifier wants to construct the values  $b_d(r_1, y)$  inductively for  $d \in [D]$ . For constructing  $b_{d+1}$  values for some  $d$ , he wants all  $u$  such that  $q_d(u) \neq 0$  (eq. (4.34)) in streaming order since he doesn't have enough space to either store the entire polynomial of degree  $n - 1$  that agrees with  $q_d$  (so as to go over all evaluations), or to parallelly evaluate it at  $n$  values while its coefficients are streamed. Hence, he asks for the following help message.

*Help message processing.* Prover continues her proof stream by sending  $\langle \hat{p}_1, Q_1, \dots, \hat{p}_D, Q_D \rangle$ , where  $Q_d := \langle \hat{q}_d(u) : u \in V \rangle$ , claiming that  $\hat{p}_d \equiv p_d$  and  $\hat{q}_d(u) = q_d(u)$  for each  $d \in [D]$  and  $u \in [n]$ .

While  $\hat{p}_d$  is streamed, Verifier computes the following in parallel:

- $\hat{p}_d(r_1, r_2)$ ;
- $p_d(r_1, r_2)$ , using eq. (4.36);
- the fingerprint  $g_d := \sum_{u \in [n]} \sum_{x \in [t]} \hat{p}_d(x, u) \beta^u$  (for some  $\beta \in_R \mathbb{F}$ ).

After reading  $\hat{p}_d$ , he checks whether  $\hat{p}_d(r_1, r_2) = p_d(r_1, r_2)$ . If so, he believes that  $\hat{p}_d \equiv p_d$  and, in turn, that  $g_d = \sum_{u \in [n]} q_d(u) \beta^u$  (by eq. (4.35)). Next, as  $Q_d$  is streamed,

- Verifier computes the fingerprint  $g'_d := \sum_{u \in [n]} \hat{q}_d(u) \beta^u$ .
- For each  $u$  with  $\hat{q}_d(u) \neq 0$ , due to eq. (4.34) (and assuming for now that the  $\hat{q}_d$  values are correct), he treats  $u$  as a stream update for  $B_{d+1}$ , and (i) maintains  $b_{d+1}(r_1, y)$  for all  $y \in [s]$ , and (ii) accordingly updates the fingerprint  $\varphi_B(\beta_1, \beta_2)$ .

After reading  $Q_d$ , he checks if the fingerprints  $g_d$  and  $g'_d$  match. If they do, he believes that all  $\hat{q}_d$  values in  $Q_d$  were correct and hence, the  $b_{d+1}$  values he constructed are correct as well. He moves on to the next iteration, i.e., starts reading  $\hat{p}_{d+1}$ .

*Final Verification.* After the  $D$ th iteration, Verifier checks if the two fingerprints  $\varphi_B(\beta_1, \beta_2)$  and  $\varphi_{\widehat{B}}(\beta_1, \beta_2)$  match. If the check passes, then he believes that the  $\widehat{\text{dist}}$  labels were correct, at least upto distance  $D$  (by Claim 4.3.21). Finally, he checks if fingerprints for  $B_D$  and  $B_{D+1}$  match to verify that vertices in  $V \setminus B_D$  are indeed unreachable.

*Error probability.* Verifier does  $O(D)$  fingerprint-checks and  $O(D)$  sum-checks, using degree- $O(n)$  polynomials. Using  $|\mathbb{F}| > n^3$  (and a union bound), the soundness error is  $< 1/n$ .

*Help and verification costs.* The set of  $\widehat{\text{dist}}$  labels sent by the Prover has size  $\widetilde{O}(n)$ . Each polynomial  $\hat{p}_d$  has  $nt$  monomials and each  $Q_d$  has  $O(n)$  field elements, and hence, size  $\widetilde{O}(n)$ . Therefore, the total hcost is  $\widetilde{O}(Dnt)$ . Initially, the  $\tilde{A}$  and  $\tilde{b}_1$  values are stored using  $\widetilde{O}(s)$  space. Next, the  $\tilde{b}_d$  and  $g_d$  values are maintained reusing space of  $b_{d-1}$  and  $g_{d-1}$  values respectively. We also use  $O(1)$  many other fingerprints that take  $O(\log n)$  space each. Hence, the total vcost is  $\widetilde{O}(s)$ .

**Theorem 4.3.22.** *There is a  $[Dnt, s]$ -scheme for unweighted SSSP, where  $D = \max_{v \in V} \text{dist}(v_s, v)$ .*

**Corollary 4.3.23.** *There is a  $[Knt, s]$ -scheme for ST-SHORTESTPATH, where  $K = \text{dist}(v_s, v_t)$ .*

*Proof.* The protocol for SSSP incurs a factor of  $D$  in the hcost since it constructs  $B_d$  for each  $d \in [D]$ . For the simpler ST-SHORTESTPATH problem, we can inductively construct balls and stop as soon as we find the destination vertex  $v_t$  in some  $B_d$  (i.e., get  $\hat{q}_{d-1}(v_t) \neq 0$ ). We must find it in  $B_K$  where  $K$  is the length of a shortest  $v_s$ - $v_t$  path. Thus, we will only incur a factor of  $K$  in the hcost, which implies a  $[Knt, s]$ -scheme for ST-SHORTESTPATH.  $\square$

Thus, we generalize the  $[Dnt, s]$ -scheme of Cormode et al. [71] from ST-SHORTESTPATH to SSSP. Our result for ST-SHORTESTPATH generalizes the  $[Kn, n]$ -scheme of Chakrabarti and Ghosh [59] by giving a smooth tradeoff and also improves upon the  $[Dnt, s]$ -scheme of Cormode et al. [71], since  $K$  can be arbitrarily smaller than  $D$ .

**Weighted SSSP Schemes.** Here, we consider the general weighted version of SSSP and give schemes for the problem in the vanilla streaming model as well as the turnstile weight update model.

**Turnstile weight update.** Assume that the edge weights are positive integers. Each stream update increments/decrements the weight of an edge. The *distance* from vertex  $u$  to vertex  $v$  refers to the weight of the shortest path from  $u$  to  $v$ . Let  $D$  be the longest distance from the source  $s$  to any other vertex reachable from it, and  $W$  be the maximum weight of an edge.

Define

$$\delta_w(X) := \prod_{\substack{w' \in [W] \\ w' \neq w}} (X - w') \bigg/ \prod_{\substack{w' \in [W] \\ w' \neq w}} (w - w').$$

Let  $A$  denote the adjacency matrix of the weighted graph  $G$ , i.e.,  $A(u, v)$  is the weight of the edge  $(u, v)$ . Let  $B_d$  (resp.  $N_d$ ) denote the set of vertices at a distance of at most (resp. exactly)  $d$  from the source vertex  $v_s$ . Then,

$$N_{d+1} = \{u \in V \setminus B_d : p_d(u) \neq 0\}, \quad (4.37)$$

$$\text{where } p_d(U) = \sum_{v \in B_d} \delta_{w(v)}(\tilde{A}(v, U)) \text{ and } w(v) = d + 1 - \text{dist}[v]. \quad (4.38)$$

*Stream processing.* Verifier chooses  $r \in_R \mathbb{F}$  and maintains  $\tilde{A}(v, r)$  for all  $v$ . He stores  $B_1$  with  $\text{dist}[v]$  labelled as 1 for each  $v \in B_1$ .

*Help message processing and verification.* Prover sends polynomials  $\hat{p}_d$  and claims that  $\hat{p}_d \equiv p_d$  for each  $d \in [D]$ . Verifier computes  $B_d$  inductively for  $d \in [D]$  as follows.

Assume that, for some  $d \in [D - 1]$ , he has the set  $B_d$  with  $\text{dist}[v]$  labeled on each vertex  $v \in B_d$ ; this holds initially as he has stored  $B_1$ . He computes  $p_d(r)$  using eq. (4.38) and checks whether  $\hat{p}_d(r) = p_d(r)$ . If the check passes, he believes that  $\hat{p}_d \equiv p_d$  and evaluates  $\hat{p}_d(u)$  for each  $u \in V \setminus B_d$  and constructs  $N_{d+1}$  using eq. (4.37). Then,  $B_{d+1}$  is given by  $N_{d+1} \uplus B_d$ .

After  $B_D$  is obtained, we get all vertices reachable from  $s$  along with their distances from  $s$ . Finally, Verifier checks if the other vertices are indeed unreachable from  $s$  by verifying that there is no cross-edge between  $B_D$  and  $V \setminus B_D$ , i.e., if  $E \cap (B_D \times (V \setminus B_D)) = \emptyset$ . (Intersection scheme, see Fact 4.1.3)

*Error probability.* Verifier uses the same element  $r$  for  $O(D)$  invocations of the sum-check protocol, where each application of the sum-check protocol is to a univariate polynomial of degree  $O(Wn)$ . Choosing  $|\mathbb{F}| > DWn^2$ , the soundness error for each invocation of the sum-check protocol is at most  $1/(Dn)$ . Taking a union bound over all  $O(D)$  invocations, we get that the total error probability of the protocol is at most  $O(1/n)$ .

*Help and verification costs* We have  $\deg p_d = O(Wn)$  for each  $d \in [D]$  and hence, hcost is  $\tilde{O}(DWn)$ . Verifier needs to store all vertices and  $\tilde{A}(v, r)$  for each  $v \in [n]$ , and hence, vcost is  $\tilde{O}(n)$ . The final disjointness can be checked by an  $[n, n]$  intersection scheme.

**Theorem 4.3.24.** *There is a  $[DWn, n]$ -scheme for SSSP in the turnstile weight update model.*

**Vanilla Stream.** We now describe a protocol for SSSP in the model where the edges arrive with their weights, without any further update on them. This is the “vanilla” streaming model.

At the end of the stream, Prover sends the distances  $\text{dist}[v]$  and  $\text{prev}[v]$ —the parent of  $v$  in the shortest path tree rooted at  $s$ —for all  $v \in V$ . Verifier checks whether the edges and their weights implied by this proof are correct, using a  $[Wn, n]$  subset scheme. Thus, if Prover is honest, we get the distance as well as shortest path from  $s$  to each vertex. But we also need to check that there is no path to any vertex shorter than the ones claimed by Prover. We describe a protocol for this.

For  $u, v \in V$  and  $w \in [W]$ , define the indicator function  $f$  as  $f(u, v, w) = 1$  iff  $A(u, v) = w$ . Let  $\tilde{f}$  be the  $\mathbb{F}$ -extension of  $f$ , for some large finite field  $\mathbb{F}$ .

Retain the definitions of  $B_d$  and  $N_d$  from last section with the definition of the polynomial  $p_d$  changed to

$$p_d(U) = \sum_{v \in B_d} \tilde{f}(v, U, d + 1 - \text{dist}_s[v]) \quad (4.39)$$

Hence, it still holds that

$$N_{d+1} = \{u \in V \setminus B_d : p_d(u) \neq 0\} . \quad (4.40)$$

*Stream processing.* The stream updates are of the form  $(u, v, w)$  denoting that  $A(u, v) = w$ . Verifier picks  $r \in_R \mathbb{F}$  and maintains  $\tilde{f}(v, r, w)$  for each  $v \in V$  and  $w \in [W]$ . He also stores the set  $B_1$  with  $\text{dist}_s$  labels set to 1 for each vertex in the set.

*Help message processing and verification.* This part is similar to the turnstile weight update protocol. Of course, this time, the Verifier computes  $p_d(r)$  using Equation (4.39).

*Error probability.* Each polynomial  $p_d$  has degree  $O(n)$ . Verifier does sum-checks for  $O(D)$  such polynomials. Choosing  $|\mathbb{F}| \gg Dn$ , we can make the error probability small by union bound.

*Help and Verification costs.* Since the degree of each  $p_d$  is at most  $n$ , the total hcost is  $\tilde{O}(Dn)$ . Verifier stores  $\tilde{f}(v, r, w)$  for each  $v \in V$  and  $w \in [W]$ , which requires  $\tilde{O}(Wn)$  space. We also need to store all vertices as we go on assigning the distance labels. Hence, the total vcost of this protocol is  $\tilde{O}(Wn)$ .

**Theorem 4.3.25.** *There is a  $[Dn, Wn]$ -scheme for SSSP in the vanilla streaming model.*

## Section 4.4

**Multipass Stream Verification**

Consider the problems `INDSETTEST` and `ST-3PATH`. The key task underlying these problems is counting the number of edges crossing between two subsets  $U$  and  $W$  of  $V$  that arrive in some adversarial streaming order along with the edges: for `INDSETTEST`,  $U$  and  $W$  are the same set; for `ST-3PATH`, they are (closed) neighborhoods of the designated vertices  $v_s$  and  $v_t$ . This is precisely the abstract problem of `CROSSEGECOUNT`. Clearly, a scheme for this problem can be used as a subroutine to solve `INDSETTEST` and `ST-3PATH`.

Any one-pass  $(h, v)$ -scheme for `CROSSEGECOUNT`, `INDSETTEST`, or `ST-3PATH` must have  $hv \geq n^2$  and hence, total cost  $h + v = \Omega(n)$ .

We therefore consider *two-pass* schemes for these problems. In particular, we design such a scheme for `CROSSEGECOUNT` with total cost  $\tilde{O}(n^{2/3})$  and apply it to obtain similar bounds for other graph problems.

We also note that our schemes can be implemented in one pass each, under natural assumptions on the way the stream is ordered; this is addressed in Section 4.4.2.

**4.4.1. One-Pass Lower Bounds**

We quickly review some relevant material from communication complexity. In the `INDEXN` problem, there are two players: Alice, who holds a vector  $\mathbf{x} \in \{0, 1\}^N$ , and Bob, who holds an index  $k \in [N]$ . Their goal is to output the bit  $\mathbf{x}_k$ . To prove lower bounds for one-pass *schemes*, we consider the *Online Merlin–Arthur* (OMA) communication model.<sup>7</sup> Here, in addition to Alice and Bob, there is a super-player, Merlin, who knows both their inputs, but is not to be blindly trusted. Merlin sends a message to Bob; then Alice sends a randomized message to Bob; finally, Bob either outputs either a bit or  $\perp$ . If Merlin is honest, Bob

<sup>7</sup>Note that our semantics are slightly different from the usual definition of Merlin–Arthur where Bob is supposed “accept” each 1-input and reject each 0-input with probability at least  $2/3$ .

should output  $\mathbf{x}_k$  with probability at least  $2/3$ ; if he is dishonest, Bob should output  $\perp$  with probability at least  $2/3$ .

The cost of an OMA protocol is the total number of bits communicated to Bob. The OMA complexity of a communication game is the minimum cost of a correct OMA protocol for it. Chakrabarti et al. [57, Theorem 3.1] showed that the OMA Complexity of  $\text{INDEX}_N$  is  $\Omega(\sqrt{N})$ . Our lower bounds follow from this result, using simple reductions from  $\text{INDEX}_N$  to the various graph problems.

Using a canonical bijection from  $[n]^2$  to  $[N]$ , Alice rewrites her input vector  $\mathbf{x} \in \{0, 1\}^N$  as a matrix  $(\mathbf{x}_{ij})_{i,j \in [n]}$ , while Bob looks at his input index  $k \in [N]$  as  $(y, z) \in [n]^2$ . Our reduction creates a graph  $G = (V, E)$  on  $2n$  vertices: the vertex set  $V$  is  $L \uplus R$  (here,  $\uplus$  denotes disjoint union), where  $|L| = |R| = n$ . We denote the  $i$ th vertex of  $L$  (resp.  $R$ ) by  $\ell_i$  (resp.  $r_i$ ). The edge set  $E$  is given by  $\{(\ell_i, r_j) : \mathbf{x}_{ij} = 1\}$ . Now, by checking if  $(\ell_y, r_z)$  is an independent set in  $G$ , or whether there's a cross-edge between the sets  $\{\ell_y\}$  and  $\{r_z\}$ , or solving  $\text{ST-3PATH}$  in the graph  $G' = (V \cup \{v_s, v_t\}, E \cup \{(v_s, \ell_y), (r_z, v_t)\})$ , Bob can solve the  $\text{INDEX}_N$  problem. Thus, a one-pass scheme that solves any of these problems must have a total cost of  $\Omega(n)$ . We remark that Fact 4.1.3 implies matching semi-streaming upper bounds for each of them.

#### 4.4.2. Two-pass Scheme for **CROSSEGE**COUNT with Applications

We now design a two-pass scheme for **CROSSEGE**COUNT, aiming for total cost  $o(n)$ .

Let  $\gamma = \gamma(U, W, G)$  denote the number of Cross-edges between  $U$  and  $W$  in a (directed or undirected) graph  $G$ . Formally, it is the number of ordered pairs  $(u, w) \in U \times W$  such that  $(u, w) \in E$ . Note that, in an undirected graph,  $\gamma$  counts an edge  $(u, w)$  with multiplicity 2 whenever  $u, w \in U \cap W$ . For some applications (e.g., counting number of 3-walks in an undirected graph), we *do* need to count them with multiplicity. We discuss later how we can remove this multiplicity if needed.

We describe a scheme that works even on turnstile graph streams, i.e., a stream of the

vertices in  $U$  and  $W$  intermixed with *updates* to edge multiplicities. Let  $L$  and  $F$  denote the characteristic vectors of the sets  $U$  and  $W$  respectively and let  $A$  be the (weighted) adjacency matrix of  $G$ . Then,

$$\gamma = \sum_{u \in U, w \in W} L_u A_{u,w} F_w. \quad (4.41)$$

Let  $t$  and  $s$  be integer parameters such that  $ts = n$ . As usual, using a canonical bijection, we represent each vertex  $v \in V$  by a pair of integers  $(x, y) \in [t] \times [s]$ . As a result, the vectors  $L, F$  transform into 2-dimensional arrays  $\ell, f$  given by  $\ell(x, y) = L_v$  and  $f(x, y) = F_v$ . As before, the adjacency matrix  $A$  turns into a 4-dimensional array  $a$ , such that  $a(x_1, y_1, x_2, y_2) = A_{v_1 v_2}$ . Let  $\tilde{\ell}, \tilde{f}$  and  $\tilde{a}$  be  $\mathbb{F}$ -extensions of  $\ell, f$  and  $a$  respectively, for a sufficiently large finite field  $\mathbb{F}$ . Now, eq. (4.41) yields

$$\gamma = \sum_{x_1, x_2 \in [t]} p(x_1, x_2), \quad \text{where} \quad (4.42)$$

$$p(X_1, X_2) = \sum_{y_1, y_2 \in [s]} \tilde{\ell}(X_1, y_1) \tilde{a}(X_1, y_1, X_2, y_2) \tilde{f}(X_2, y_2). \quad (4.43)$$

For  $i \in \{1, 2\}$ ,  $\deg_{X_i} p = 2t - 2$ . Thus, it follows that the number of monomials in  $p$  is at most  $O(t^2)$ , and the total degree of  $p$  is  $O(t)$ .

We are now ready to design a two-pass scheme for **CROSSEDGECOUNT**.

*Stream processing.* Verifier first chooses  $r_1, r_2 \in_R \mathbb{F}$ . For  $y \in [s]$ , define

$$g(y) := \sum_{y' \in [s]} \tilde{a}(r_1, y, r_2, y') \tilde{f}(r_2, y') \quad (4.44)$$

Thus,

$$p(r_1, r_2) = \sum_{y \in [s]} \tilde{\ell}(r_1, y) g(y). \quad (4.45)$$

**Pass 1.** Only process the vertices in  $L$  and  $F$  in the stream. Maintain (using Fact 4.1.1) two  $s$ -dimensional vectors:  $\tilde{\ell}(r_1, y)$  and  $\tilde{f}(r_2, y)$ , where  $y \in [s]$ .

**Pass 2.** Only process the edges in the stream. We want to maintain the  $s$ -dimensional vector  $g(y)$  so that we can compute  $p(r_1, r_2)$  using eq. (4.45). Suppose that the  $j$ th edge update  $(x_1, y_1, x_2, y_2)_j$  adds  $\Delta_j$  to that edge's multiplicity. This results in updates to several entries of  $\tilde{a}$ , but we want to use only  $O(s)$  space, so we cannot afford to maintain  $\tilde{a}$  directly. Instead, for each  $j \in [m]$ , let  $g_j$  and  $\tilde{a}_j$  denote the values of  $g$  and  $\tilde{a}$  (respectively) after the  $j$ th stream update. Then

$$\begin{aligned} g_j(y) &= \sum_{y' \in [s]} \tilde{f}(r_2, y') \tilde{a}_j(r_1, y, r_2, y') \\ &= \sum_{y' \in [s]} \tilde{f}(r_2, y') (\tilde{a}_{j-1}(r_1, y, r_2, y') + \Delta_j \delta_{(x_1, y_1, x_2, y_2)_j}(r_1, y, r_2, y')) \quad (4.46) \\ &= g_{j-1}(y) + h_j(y), \end{aligned}$$

where eq. (4.46) follows from eq. (4.2) and

$$h_j(y) := \sum_{y' \in [s]} \tilde{f}(r_2, y') \Delta_j \delta_{(x_1, y_1, x_2, y_2)_j}(r_1, y, r_2, y'). \quad (4.47)$$

Hence, after the  $j$ th update, the Verifier can compute  $h_j(y)$  and maintain the vector  $g(y)$ .

*Help message.* After the second pass, Prover sends a polynomial  $\hat{p}(X_1, X_2)$  (as a stream of coefficients) that she claims equals  $p(X_1, X_2)$ .

*Verification and output.* At the end of the second pass, Verifier gets  $g(y)_m = g(y)$  for each  $y$ . Now, he uses eq. (4.45) to compute the check value  $p(r_1, r_2)$  and the result value  $\hat{\gamma} := \sum_{x_1, x_2 \in [t]} \hat{p}(x_1, x_2)$ . If he finds that  $p(r_1, r_2) \neq \hat{p}(r_1, r_2)$ , he outputs  $\perp$ . Otherwise, he believes that  $\hat{p} \equiv p$  and exploiting eq. (4.42), outputs  $\hat{\gamma}$  as the answer.

Now, we analyze the correctness and complexity parameters of the scheme.

*Error probability.* The protocol errs only when  $\hat{p} \neq p$ , but Verifier's check passes. Then,

$(r_1, r_2) \in \mathbb{F}^2$  must be a root of the nonzero polynomial  $\hat{p} - p$ . We noted that its total degree is  $O(t)$ . Thus, the Schwartz-Zippel Lemma bounds the error probability by at most  $O(t)/|\mathbb{F}| < 1/n$ , for large enough choice of  $|\mathbb{F}|$ .

*Help and Verification costs.* The polynomial  $\hat{p}$  has  $O(t^2)$  monomials, and so, the hcost is  $\tilde{O}(t^2)$ . Verifier stores constant many vectors of size  $s$  at a time and incurs a vcost of  $\tilde{O}(s)$ .

Thus, we obtain a two-pass  $[t^2, s]$ -scheme for **CROSSEGECOUNT**, for parameters  $t, s$  with  $ts = n$ . Setting  $t = n^{1/3}$  and  $s = n^{2/3}$ , we get a scheme with total cost  $\tilde{O}(n^{2/3})$ .

Finally, we discuss how one can count cross-edges between  $U$  and  $W$  when they are defined as unordered pairs. Define this problem as **CROSSEGECOUNT-UNIQ**. Let  $\gamma'$  be the number of edges that  $\gamma$  counts with multiplicity 2, i.e., the number of undirected edges  $(u, w) \in U \times W$  such that  $u, w \in U \cap W$ . Then,

$$\gamma' = \sum_{u \in U, w \in W} L_u F_u A_{u,w} L_w F_w. \quad (4.48)$$

Hence, we modify the definitions of  $p(X_1, X_2)$  and  $g(y)$  as

$$p(X_1, X_2) := \sum_{y_1, y_2 \in [s]} \tilde{\ell}(X_1, y_1) \tilde{f}(X_1, y_1) \tilde{a}(X_1, y_1, X_2, y_2) \tilde{\ell}(X_2, y_2) \tilde{f}(X_2, y_2). \quad (4.49)$$

$$g(y) := \sum_{y' \in [s]} \tilde{a}(r_1, y, r_2, y') \tilde{\ell}(r_2, y') \tilde{f}(r_2, y'). \quad (4.50)$$

Then, proceeding as in **CROSSEGECOUNT**, we compute  $\gamma'$ . Thus, we can compute  $\gamma$  and  $\gamma'$  in parallel and finally output  $\gamma - \gamma'$  as the answer to **CROSSEGECOUNT-UNIQ**.

**Theorem 4.4.1.** *For parameters  $t, s$  with  $ts = n$ , there are two-pass  $[t^2, s]$ -schemes for CROSSEDGECOUNT and CROSSEDGECOUNT-UNIQ. In particular, there are two-pass schemes with total cost  $\tilde{O}(n^{2/3})$ .  $\square$*

**Applications.** Our scheme for CROSSEDGECOUNT can be used as a black box for solving a number of other problems. These include standard problems like INDSETTEST and ST-3PATH, as well as their generalizations or variations such as the following problems.

- **INDUCEDGECOUNT:** Given a graph  $G = (V, E)$  and a subset  $U$  of  $V$ , find the number of edges in  $G$  that are induced by  $U$ .
- **ROOTEDTRIANGLECOUNT:** Given a (directed or undirected) graph  $G = (V, E)$  and a vertex  $v_r \in V$ , find the number of triangles in  $G$  that are rooted at  $v_r$ .

**Corollary 4.4.2.** *Let  $t$  and  $s$  be parameters such that  $ts = n$ . Then each of the problems INDUCEDGECOUNT, INDSETTEST, ST-3PATH, and ROOTEDTRIANGLECOUNT admits a two-pass  $[t^2, s]$ -scheme; in particular, each of them admits a two-pass scheme with total cost  $\tilde{O}(n^{2/3})$ .*

*Proof.* For INDUCEDGECOUNT, if the input graph is undirected, then considering  $U$  and  $W$  as the same set, solve CROSSEDGECOUNT-UNIQ. (Alternatively, solve CROSSEDGECOUNT and divide the answer by two.) If the graph is directed, then solve CROSSEDGECOUNT.

For INDSETTEST, solve INDUCEDGECOUNT on  $U$  and check whether the answer equals zero.

For ST-3PATH, use a scheme for CROSSEDGECOUNT to find the number of cross-edges between the closed neighborhoods  $N[v_s]$  and  $N[v_t]$  of vertices  $v_s$  and  $v_t$ . This actually solves the more general problem of counting the number of walks of length at most 3 from  $v_s$  to  $v_t$ . Checking whether this number is non-zero decides ST-3PATH.

Finally, for ROOTEDTRIANGLECOUNT, if the input graph is undirected, solve INDUCEDEDGECOUNT on  $N(v_r)$ . Otherwise, solve CROSSEGECOUNT on the out-neighborhood  $N^+(v_r)$  and in-neighborhood  $N^-(v_r)$  of  $v_r$ .  $\square$

**One-Pass Schemes for Certain Stream Orderings.** Our two-pass solution to the CROSSEGECOUNT problem, as well as its corollaries, allowed the vertices and edge updates to be arbitrarily intermixed in the input stream. That said, it is interesting to focus on a natural restriction of these problems where the vertices are streamed first, followed by the edge updates. For the ST-3PATH problem, the corresponding restriction is that the edges incident to  $v_s$  and  $v_t$  appear before any other edges in the stream; for ROOTEDTRIANGLECOUNT, it is that the edges incident to  $v_r$  appear first.

Under such a restriction on the stream ordering, our two-pass solutions naturally become one-pass, as we now note.

**Proposition 4.4.3.** *The schemes for CROSSEGECOUNT and CROSSEGECOUNT-UNIQ in Theorem 4.4.1 and for INDUCEDEDGECOUNT, INDSETTEST, ST-3PATH, and ROOTEDTRIANGLECOUNT in Corollary 4.4.2 can each be implemented in one pass under a restricted stream ordering as noted above.*

*Proof.* Consider the protocol described in Section 4.4.2. Note that the first pass processes only vertices and the second pass processes only edges. This implies the claimed results for CROSSEGECOUNT, CROSSEGECOUNT-UNIQ, INDUCEDEDGECOUNT, and INDSETTEST. For ST-3PATH, note that requiring edges incident to  $v_s$  and  $v_t$  to arrive first is equivalent to the vertex sets  $N(v_s)$  and  $N(v_t)$  arriving first. A similar consideration applies to ROOTEDTRIANGLECOUNT.  $\square$

It is important to note that despite the restriction on the stream ordering, the schemes in Proposition 4.4.3 are nontrivial. Without Prover's help, the problems remain hard, even with multiple passes. We give the simple proof for the basic problem CROSSEGECOUNT.

**Proposition 4.4.4.** *Any  $p$ -pass streaming algorithm for CROSSEGECOUNT, with vertices streamed before edges, requires  $\Omega(n/p)$  space, even for insertion-only streams.*

*Proof.* We reduce from  $\text{DISJ}_n$ , the set-disjointness communication problem on the universe  $[n]$ . Recall that, in  $\text{DISJ}_n$ , Alice holds a set  $x \subseteq [n]$  and Bob holds a set  $y \subseteq [n]$ . Their goal is to determine whether or not  $x \cap y = \emptyset$ . This problem has randomized communication complexity  $R(\text{DISJ}_n) = \Omega(n)$  [153].

Consider an  $(n + 1)$ -vertex graph  $G$  where  $V(G) = \{0, \dots, n\}$  and  $E(G) = \{\{0, i\} : i \in y\}$ . Let  $U = \{0\}$  and  $W = x$ . Then the number of cross edges in  $G$  from  $U$  to  $W$  is non-zero iff  $x \cap y \neq \emptyset$ . The result now follows along standard lines.  $\square$

#### 4.4.3. A Multi-Pass Scheme for Detecting Short Paths

In Section 4.4, we obtained a scheme for ST-3PATH of total cost  $o(n)$  using two passes over the input. We investigate if the same is true for ST-KPATH (for  $k > 3$ ) if we allow “a few” more passes. For constant  $k$ , we answer this in the affirmative as we generalize the scheme for ST-3PATH and obtain such a scheme for ST-KPATH with  $\lceil k/2 \rceil$  passes.

As usual,  $A$  denotes the adjacency matrix of the multigraph  $G$ . Let  $L$  and  $F$  be the characteristic vectors of  $N[v_s]$  and  $N[v_t]$  respectively. Let  $\kappa = \kappa(G)$  denote the number of walks of length at most  $k$  from  $v_s$  to  $v_t$  in  $G$ . Then,

$$\kappa = \sum_{u_1, \dots, u_{k-1} \in V} L_{u_1} \left( \prod_{i=1}^{k-2} A_{u_i, u_{i+1}} \right) F_{u_{k-1}}. \quad (4.51)$$

Note that there is a path of length at most  $k$  from  $v_s$  to  $v_t$  iff  $\kappa > 0$ . Therefore, computing  $\kappa$  suffices.

Let  $h$  and  $v$  be integer parameters with  $hv = n$ . Again, using a canonical bijection, we represent each vertex  $u \in V$  by a pair of integers  $(x, y) \in [h] \times [v]$ . The vectors  $L$  and  $F$  become 2-dimensional arrays  $\ell$  and  $f$ , given by  $\ell(x, y) = L_u$  and  $f(x, y) = F_u$ . Again, the adjacency matrix  $A$  turns into a 4-dimensional array  $a$ , such that  $a(x, y, x', y') = A_{uu'}$ . Let

$\tilde{\ell}$ ,  $\tilde{f}$ , and  $\tilde{a}$  be  $\mathbb{F}$ -extensions of  $\ell$ ,  $f$ , and  $a$  respectively, where  $\mathbb{F}$  is a finite field of cardinality  $q$  and  $q$  is a prime chosen *uniformly at random* from  $[n^3, n^4]$ . Then eq. (4.51) gives

$$\kappa = \sum_{x_1, \dots, x_{k-1} \in [h]} p(x_1, \dots, x_{k-1}), \quad \text{where} \quad (4.52)$$

$$p(X_1, \dots, X_{k-1}) = \sum_{y_1, y_2 \in [v]} \tilde{\ell}(X_1, y_1) \left( \prod_{i=1}^{k-2} \tilde{a}(X_i, y_i, X_{i+1}, y_{i+1}) \right) \tilde{f}(X_{k-1}, y_{k-1}). \quad (4.53)$$

For  $i \in [k-1]$ ,  $\deg_{X_i} p = 2h - 2$ . Therefore, the number of monomials in  $p$  is at most  $O(h^{k-1})$  and the total degree is  $O(kh)$ .

We present a  $\lceil k/2 \rceil$ -pass protocol for ST-KPATH.

*Stream processing.* Verifier chooses  $r_1, \dots, r_{k-1} \in_R \mathbb{F}$ .

**Pass 1.** Process only the vertices in  $N_1[v_s]$  and  $N_1(v_t)$  in the stream. We maintain, for each  $y \in [v]$ , two vectors of size  $v$ :  $\tilde{\ell}(r_1, y)$  and  $\tilde{f}(r_{k-1}, y)$ , where  $y \in [s]$ .

**Pass  $i$ , for  $2 \leq i \leq \lceil k/2 \rceil$ .** Define  $g_0(y) := \tilde{\ell}(r_1, y)$  and  $g_k(y) = \tilde{f}(r_{k-1}, y)$ . For each  $y \in [v]$ , compute  $g_{i-1}(y) := \sum_{y' \in [v]} \tilde{a}(r_{i-1}, y, r_i, y') g_{i-2}(y')$  as well as  $g_{k-i+1}(y) := \sum_{y' \in [v]} \tilde{a}(r_{k-i}, y, r_{k-i+1}, y') g_{k-i+2}(y')$ . The  $g_j(y)$  values are updated dynamically with the stream updates in a similar way as in the protocol for CROSSEDGE-COUNT in Section 4.4.2.

*Help message.* At the end of the final pass, Prover sends a polynomial  $\hat{p}(X_1, \dots, X_{k-1})$ —as a stream of coefficients—that she claims equals  $p(X_1, \dots, X_{k-1})$ .

*Verification and output.* After the final pass, Verifier computes  $\sum_{y \in [v]} g_{\lceil k/2 \rceil}(y) g_{\lceil k/2 \rceil + 1}(y)$ , which, by Equation (4.53), equals  $p(r_1, \dots, r_{k-1})$ . If he finds that it doesn't equal  $\hat{p}(r_1, \dots, r_{k-1})$ , he outputs  $\perp$ . Otherwise, he believes that  $\hat{p} \equiv p$  and, following eq. (4.52), computes  $\hat{\kappa} := \sum_{x_1, \dots, x_{k-1} \in [h]} \hat{p}(x_1, \dots, x_{k-1})$ . He outputs YES if  $\hat{\kappa} > 0$

and NO otherwise.

*Error probability.* We err when  $\hat{p} \neq p$ , but Verifier's check passes. In this case,  $(r_1, \dots, r_{k-1}) \in \mathbb{F}^{k-1}$  is a root of the nonzero polynomial  $\hat{p} - p$ . We noted that its total degree is at most  $O(kh)$ . By the Schwartz-Zippel Lemma (Fact 4.1.2), the probability of this event is at most  $O(kh)/|\mathbb{F}| < 1/n$ . We err also when  $\hat{\kappa}$  is non-zero, but the prime  $q$  divides  $\hat{\kappa}$ , making  $\hat{\kappa} \bmod q = 0$ . But  $\hat{\kappa}$  can have value at most  $2^n n!$ , and so has at most  $O(n \log n)$  distinct prime factors. Since we chose  $q$  uniformly at random within  $[n^3, n^4]$ , by the Prime Number Theorem, the probability that  $q$  equals one of the prime factors of  $\hat{\kappa}$  is at most  $1/n^2$ . Hence, the total error is at most  $1/n + 1/n^2$ .

*Help and Verification costs.* The number of monomials of  $\hat{p}$  is  $O(h^{k-1})$ , giving an hcost of  $\tilde{O}(h^{k-1})$ . Verifier reuses space and, during each pass, stores  $O(1)$  many  $v$ -dimensional vectors, each entry of which is  $O(\log n)$  bits long. Thus, the vcost is  $\tilde{O}(v)$ .

This gives a  $\lceil k/2 \rceil$ -pass  $[h^{k-1}, v]$ -scheme for ST-KPATH, for parameters  $h, v$  with  $hv = n$ . Setting  $h = n^{1/k}$  and  $v = n^{1-1/k}$ , we get a scheme with total cost  $\tilde{O}(n^{1-1/k})$ .

**Theorem 4.4.5.** *There is a  $\lceil k/2 \rceil$ -pass  $[n^{1-1/k}, n^{1-1/k}]$ -scheme for ST-KPATHCOUNT in a (directed or undirected) multigraph. In particular, for constant  $k$ , there is constant-pass scheme with total cost  $o(n)$ .  $\square$*

We note the contrast between this result and that of Guruswami and Onak [95]. They showed a lower bound of  $\Omega(n^{1+\Omega(1/k)}/k^{O(1)})$  for ST-KPATH in  $k/2 - 1$  passes in the basic (sans prover) streaming model (for even  $k$ ). Our results show that using  $\lceil k/2 \rceil$  passes, we can obtain a scheme for the same problem with total cost of  $\tilde{O}(n^{1-1/k})$ .

---

## Chapter 5

---

# Conclusions and Future Directions

In this thesis, we resolved some important questions about the space complexity of several graph problems in multiple variants of the data streaming model. In Chapter 2, we studied the most popular and well-studied variant that we call the classical or standard streaming model. In Section 2.1, we carried out an investigation on *directed graph* problems in this model. The main message from our investigation was that while many fundamental digraph problems such as topological sorting and feedback arc set are hard for general streaming digraphs even when the stream is randomly ordered, they turn out to have interesting and efficient algorithms for the important and well-studied class of tournament graphs. In Section 2.2, we designed a graph coloring framework with the number of colors parameterized by the graph degeneracy  $\kappa$  and showed that for a large class of graphs, it gives significantly more color-efficient algorithms than any  $(\Delta + 1)$ -coloring algorithm not only in the streaming model, but also in graph query and distributed models of computation such as MPC, Congested Clique, and LOCAL, where graph coloring is an extensively studied problem. We also showed that while any graph is both  $(\Delta + 1)$ - and  $(\kappa + 1)$ -colorable, the space complexity of attaining such colorings vary significantly in the streaming model: a  $(\Delta + 1)$ -coloring is achievable in semi-streaming space but a  $(\kappa + 1)$ -coloring is not possible in sublinear space. In Chapter 3, we considered the adversarially robust streaming setting

and showed that graph coloring is harder in this model even with respect to  $\Delta$ : fixing the number of colors to  $O(\Delta)$ , there is a quadratic gap in space complexity, while fixing the space to semi-streaming, we have a quadratic gap for number of colors. This is the first separation between classical and robust streaming for a natural problem. In Chapter 4, we explored streaming verification in the annotated streaming setting. In Section 4.2, we designed two new verification schemes for the fundamental problem of computing frequency-based functions: the schemes are much simpler than the state of the art, and in addition, improve upon the space and communications cost. This yields improved subroutines for many graph problems as frequency-functions have wide applications in graph streaming. Finally, in Section 4.3, we designed efficient schemes for a variety of graph problems: some of them remarkably improve upon the state of the art, some of them were rather unexpected, while some of them achieve smooth optimal tradeoffs between space usage and proof size, thus settling the complexity of the respective problems in broad regimes.

We conclude by discussing some open questions and future research directions that emanate from the results in this thesis.

**Stronger lower bounds for vertex-ordering problems.** In classical streaming, we showed that similar to the case of adversarial stream order, problems like STCONN-DAG, ACYC, and TOPO-SORT require roughly  $\Omega(n^{1+1/p})$  space for  $p$  passes on random-order streams. However, there is no known upper bound anywhere close to these lower bounds. In particular, the question *is there an  $O(\log n)$ -pass semi-streaming algorithm for any of these problems* (which is not ruled out by the existing lower bounds) is still open. In fact, it is open for the random-order model, even with error probability loosened to  $1/p^p$  (rather than constant for any  $p$ ). In other words, a matching upper bound for Theorem 2.1.9 is unknown.

We believe such upper bounds are unknown because they do not exist, at least for the harder adversarial order version. Concretely, we conjecture that  *$s$ - $t$  connectivity does not admit any  $O(\text{polylog}(n))$ -pass semi-streaming algorithm*. As discussed in Section 2.1.9,

there has been some progress on multipass lower bounds for this problem, albeit only for  $O(\sqrt{\log n})$  passes.

**Settling the streaming complexity of degeneracy-based coloring.** Theorem 2.2.15 shows that coloring in semi-streaming space requires  $\kappa + \tilde{\Omega}(\sqrt{\kappa})$  colors. Our semi-streaming algorithm achieves  $\kappa + \kappa/\text{polylog}(n)$  colors. Closing this gap remains an open problem: *find  $\lambda = \lambda(\kappa) \in [\sqrt{\kappa}, \kappa]$  such for graph coloring in semi-streaming space,  $\kappa + \tilde{\Theta}(\lambda)$  colors are necessary and sufficient.* More generally, *can we achieve a smooth and tight color-space tradeoff to fully settle the one-pass streaming complexity of  $\kappa$ -based coloring?* Note that improving upon our semi-streaming upper bound would require fairly new techniques since we showed that using only palette sparsification techniques, our bound is the best that one can get. Further, another interesting future direction is considering the problem in the multipass setting so as to improve upon the number of colors. In particular, *how many passes do we need to obtain a  $(\kappa + 1)$ -coloring?*

**Robust algorithms for graph streams with deletions.** The recent surge of interest in adversarially robust streaming has developed its literature considerably over the last couple of years; yet graph problems remained mostly unexplored in this setting until our work. We observed that this might be due to the fact that a large number of graph problems have streaming algorithms that are either already robust or can be “robustified” incurring only a slight increase in space by using known general frameworks. But these frameworks apply for *insertion-only* streams; when deletions are allowed, we still do not know efficient robust streaming algorithms for many graph problems. An interesting direction would be to explore whether there exist efficient robust dynamic streaming algorithms for well-studied graph problems (such as triangle counting, densest subgraphs), which have efficient algorithms in the classical setting even when deletions are allowed. In fact, our robust coloring algorithms for insert-delete streams use sublinear space only when the stream length is  $O(n \cdot \text{poly}(\Delta))$ ; they are inefficient for longer streams. We ask *can we design a ro-*

*bust  $\text{poly}(\Delta)$ -coloring  $o(n\Delta)$ -space algorithm for dynamic graph streams of any length?*

Whether efficient robust algorithms for turnstile streams can be obtained is open even for classical data streaming problems such as *frequency moments* and other statistical estimation problems on which most of the robust streaming literature has focused. In fact, it is believed that there is a large separation between insert-only and turnstile streams for these problems in the robust setting; proving such a separation is one of the major open questions in the area.

**Settling the annotated-streaming complexity of frequency-based functions.** An important open problem in the field of stream verification is to determine the asymptotic complexity of the general problem of computing frequency-based functions. Chakrabarti et al. [57] showed that any online or prescient  $(h, v)$ -scheme for the problem requires  $hv \geq n$ . Note that this lower bound leaves open the possibility of a  $(\sqrt{n}, \sqrt{n})$ -scheme, while the best known scheme achieves  $(\tilde{O}(n^{2/3}), \tilde{O}(n^{2/3}))$  for both online and prescient settings. Can we match the lower bound (up to polylogarithmic factors) and get an  $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$ -scheme for the problem, even if prescient? What about for even special cases like  $F_0$  or  $F_\infty$ ? Recall that there do exist such online schemes for the  $k$ th frequency moment for any constant  $k \in \mathbb{Z}^+$  [57]. Also, it is possible to get such a scheme for  $F_0$  if we allow multiple rounds of interaction [94]. Any strict improvement on the lower bound would be extremely interesting and a breakthrough. Currently, we don't know of a function in the turnstile streaming model for which any online  $(h, v)$ -scheme must have total cost  $h + v \geq \omega(\sqrt{N})$  where  $N$  is the lower bound on its classical streaming complexity. This is related to the major open question of breaking the “ $\sqrt{N}$  barrier” for the Merlin-Arthur (MA) communication model.

**Fully settling the annotated-streaming complexities of TRIANGLECOUNT and MAX-MATCHING.** We obtained a  $[t, ns]$ -scheme for TRIANGLECOUNT for any  $t, s$  satisfying  $ts = n$ . It matches the lower bound  $hv \geq n^2$  for any  $(h, v)$ -scheme [57]. Thus, for the regime of  $(\text{hcost} \leq n, \text{vcost} \geq n)$ , which we call the *laconic* regime, the complexity of

the triangle counting problem is settled. However, for the opposite regime of ( $\text{hcost} \geq n$ ,  $\text{vcost} \leq n$ ) that we call the *frugal* regime, the best we have is an  $[nt^2, s]$ -scheme, which is not known to be optimal. We ask if we can match the lower bound for this regime as well, and fully settle the complexity of triangle counting in the annotated streaming model: *does there exist  $[nt, s]$ -schemes for triangle counting for all  $t, s$  satisfying  $ts = n$ ?* We conjecture that the answer is yes since it does hold for the settings  $(t = n, s = 1)$  and  $(t = 1, s = n)$ . Also, there does not seem to be an information theoretic bottleneck that prevents the smooth tradeoff.

For any  $t, s$  satisfying  $ts = n$ , we have an  $[nt, s]$ -scheme for MAXMATCHING, thus matching the lower bound of  $hv \geq n^2$  for any  $[h, v]$ -scheme. It settles the complexity of the problem in the frugal regime. Now, for the opposite laconic regime, we do not know any scheme for the general problem, let alone optimal. The barrier seems to be that a natural “witness” for the problem is an actual maximum matching of the graph, which can be of size  $\Theta(n)$ , making the  $\text{hcost} \Omega(n)$ . In fact, we showed that large matching size is indeed the sole barrier to obtaining a laconic scheme for the problem: for a graph with max-matching size  $\alpha'$ , we gave an  $[\alpha', n^2/\alpha']$ -scheme, which is laconic when  $\alpha' = o(n)$ . Hence, we ask the following question: *does there exist an  $[o(n), o(n^2)]$ -scheme for MAXMATCHING?* We conjecture that the answer is negative. The barrier of a  $\Theta(n)$ -length proof seems inherent. If we can prove this, it would imply that our upper bounds for MAXMATCHING are essentially optimal for any regime, which would fully settle its complexity in the annotated streaming model.

There are several other problems for which we get optimal frugal schemes but are unable to get any laconic scheme because a natural witness has size  $\Theta(n)$ . These include MIS, TOPO-SORT, and ACYC, for which natural proofs are, respectively, a valid maximal independent set, a valid topological ordering, and a cycle (for the NO case) or a topological ordering (for the YES case). We conjecture that all these problems belong to the same class

as MAXMATCHING: there are no laconic schemes for these problems for general graphs.

---

# Bibliography

- [1] Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz, *Smaller cuts, higher lower bounds*, CoRR **abs/1901.01630** (2019).
- [2] Amirali Abdullah, Samira Daruki, Chitradeep Dutta Roy, and Suresh Venkatasubramanian, *Streaming verification of graph properties*, Proc. 27th International Symposium on Algorithms and Computation, 2016, pp. 3:1–3:14.
- [3] Farid Ablayev, *Lower bounds for one-way probabilistic communication complexity and their application to space complexity*, Theor. Comput. Sci. **175** (1996), no. 2, 139–159.
- [4] Kook Jin Ahn and Sudipto Guha, *Graph sparsification in the semi-streaming model*, Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II, Lecture Notes in Computer Science, vol. 5556, Springer, 2009, pp. 328–338.
- [5] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor, *Analyzing graph structure via linear measurements*, Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, 2012, pp. 459–467.
- [6] Nir Ailon, *Active learning ranking from pairwise preferences with almost optimal query complexity*, Advances in Neural Information Processing Systems 24

- (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2011, pp. 810–818.
- [7] Nir Ailon, Moses Charikar, and Alantha Newman, *Aggregating inconsistent information: Ranking and clustering*, J. ACM **55** (2008), no. 5, 23:1–23:27.
- [8] Noga Alon and Sepehr Assadi, *Palette sparsification beyond  $(\Delta+1)$  vertex coloring*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference, LIPIcs, vol. 176, 2020, pp. 6:1–6:22.
- [9] Noga Alon, László Babai, and Alon Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, Journal of algorithms **7** (1986), no. 4, 567–583.
- [10] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, J. Comput. Syst. Sci. **58** (1999), no. 1, 137–147, Preliminary version in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.
- [11] Stanislav Angelov, Keshav Kunal, and Andrew McGregor, *Sorting and selection with random costs*, LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings, 2008, pp. 48–59.
- [12] Sabeur Aridhi, Martin Brugnara, Alberto Montresor, and Yannis Velegrakis, *Distributed  $k$ -core decomposition and maintenance in large dynamic graphs*, Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016, 2016, pp. 161–168.
- [13] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM **45** (1998),

- no. 3, 501–555, Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.
- [14] Sanjeev Arora and Shmuel Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM **45** (1998), no. 1, 70–122, Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.
- [15] Sepehr Assadi, *Sublinear algorithms for  $(\Delta + 1)$  vertex coloring*, Lecture at Sublinear Algorithms and Nearest-Neighbor Search Workshop, Simons Institute; available online at [https://www.youtube.com/watch?v=VU7Y\\_8ZcNu0&t=2206](https://www.youtube.com/watch?v=VU7Y_8ZcNu0&t=2206), 2018.
- [16] Sepehr Assadi, Andrew Chen, and Glenn Sun, *Deterministic graph coloring in the streaming model*, arXiv preprint arXiv:2109.14891 (2021), To appear in STOC 2022.
- [17] Sepehr Assadi, Yu Chen, and Sanjeev Khanna, *Sublinear algorithms for  $(\Delta + 1)$  vertex coloring*, Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms, 2019, pp. 767–786.
- [18] Sepehr Assadi, Sanjeev Khanna, and Yang Li, *On estimating maximum matching size in graph streams*, Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms, 2017, pp. 1723–1742.
- [19] Sepehr Assadi, Pankaj Kumar, and Parth Mittal, *Brooks’ theorem in graph streams: A single-pass semi-streaming algorithm for  $\Delta$ -coloring*, CoRR **abs/2203.10984** (2022), To appear in STOC 2022.
- [20] Sepehr Assadi and Ran Raz, *Near-quadratic lower bounds for two-pass graph streaming algorithms*, 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020, IEEE, 2020, pp. 342–353.

- [21] Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer, *A framework for adversarial streaming via differential privacy and difference estimators*, CoRR **abs/2107.14527** (2021).
- [22] Gary D. Bader and Christopher WV Hogue, *An automated method for finding molecular complexes in large protein interaction networks*, BMC Bioinformatics **4** (2003), no. 1, 2.
- [23] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii, *Densest subgraph in streaming and mapreduce*, International Conference on Very Large Data Bases **5** (2012), no. 5, 454–465.
- [24] Balabhaskar Balasundaram and Sergiy Butenko, *Graph domination, coloring and cliques in telecommunications*, Handbook of Optimization in Telecommunications, Springer, 2006, pp. 865–890.
- [25] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan, *Counting distinct elements in a data stream*, Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science, 2002, pp. 128–137.
- [26] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 623–632.
- [27] Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot, *Dynamic graph coloring*, Workshop on Algorithms and Data Structures, 2017, pp. 97–108.
- [28] Leonid Barenboim, *Deterministic  $(\Delta + 1)$ -coloring in sublinear (in  $\Delta$ ) time in static, dynamic, and faulty networks*, Journal of the ACM (JACM) **63** (2016), no. 5, 47.

- 
- [29] Leonid Barenboim and Michael Elkin, *Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition*, Distributed Computing **22** (2010), no. 5-6, 363–379.
- [30] Leonid Barenboim and Michael Elkin, *Deterministic distributed vertex coloring in polylogarithmic time*, Journal of the ACM (JACM) **58** (2011), no. 5, 23.
- [31] Leonid Barenboim and Michael Elkin, *Distributed graph coloring: Fundamentals and recent developments*, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2013.
- [32] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider, *The locality of distributed symmetry breaking*, Journal of the ACM (JACM) **63** (2016), no. 3, 20.
- [33] Nicolas Barnier and Pascal Brisset, *Graph coloring for air traffic flow management*, Annals of operations research **130** (2004), no. 1-4, 163–178.
- [34] MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni, *Almost optimal streaming algorithms for coverage problems*, Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017, 2017, pp. 13–23.
- [35] Anubhav Baweja, Justin Jia, and David P. Woodruff, *An efficient semi-streaming PTAS for tournament feedback arc set with few passes*, 13th Innovations in Theoretical Computer Science Conference, ITCS 2022, LIPIcs, vol. 215, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 16:1–16:23.
- [36] Paul Beame, Paraschos Koutris, and Dan Suciu, *Communication steps for parallel query processing*, J. ACM **64** (2017), no. 6, 40:1–40:58.
- [37] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis, *Efficient semi-streaming algorithms for local triangle counting in massive graphs*, Proceedings

- of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008, ACM, 2008, pp. 16–24.
- [38] Soheil Behnezhad, Mahsa Derakhshan, and Mohammad Taghi Hajiaghayi, *Brief announcement: Semi-mapreduce meets congested clique*, CoRR **abs/1802.10297** (2018).
- [39] Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak, *Adversarially robust streaming via dense-sparse trade-offs*, Symposium on Simplicity in Algorithms (SOSA), 2022, pp. 214–227.
- [40] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev, *A framework for adversarially robust streaming algorithms*, Proc. 39th ACM Symposium on Principles of Database Systems, 2020, p. 63–80.
- [41] Omri Ben-Eliezer and Eylon Yogev, *The adversarial robustness of sampling*, Proc. 39th ACM Symposium on Principles of Database Systems, ACM, 2020, pp. 49–62.
- [42] Suman K. Bera and Amit Chakrabarti, *Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams*, 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017), 2017, pp. 11:1–11:14.
- [43] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh, *Graph coloring via degeneracy in streaming and other space-conscious models*, 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference), LIPIcs, vol. 168, 2020, pp. 11:1–11:21.
- [44] Suman Kalyan Bera and Prantar Ghosh, *Coloring in graph streams*, CoRR **abs/1807.07640** (2018).

- [45] Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana, *Even the easiest(?) graph coloring problem is not easy in streaming!*, 12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference, LIPIcs, vol. 185, 2021, pp. 15:1–15:19.
- [46] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai, *Dynamic algorithms for graph coloring*, Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, SIAM, 2018, pp. 1–20.
- [47] Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon, *Fully dynamic  $(\Delta+1)$ -coloring in constant update time*, CoRR **abs/1910.02063** (2019).
- [48] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, *Preventing unraveling in social networks: The anchored  $k$ -core problem*, SIAM Journal on Discrete Mathematics **29** (2015), no. 3, 1452–1475.
- [49] J.A. Bondy and U.S.R Murty, *Graph theory*, 1st ed., Springer Publishing Company, Incorporated, 2008.
- [50] Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou, *Adversarial robustness of streaming algorithms through importance sampling*, Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, 2021, pp. 3544–3557.
- [51] Harry Buhrman and Ronald de Wolf, *Complexity measures and decision tree complexity: a survey*, Theor. Comput. Sci. **288** (2002), no. 1, 21–43.

- [52] Gregory J Chaitin, *Register allocation & spilling via graph coloring*, ACM Sigplan Notices, vol. 17, 1982, pp. 98–105.
- [53] Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein, *Register allocation via coloring*, Computer languages **6** (1981), no. 1, 47–57.
- [54] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler, *Annotations for sparse data streams*, Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms, 2014, pp. 687–706.
- [55] Amit Chakrabarti, Graham Cormode, and Andrew McGregor, *Annotations in data streams*, Proc. 36th International Colloquium on Automata, Languages and Programming, 2009, pp. 222–234.
- [56] Amit Chakrabarti, Graham Cormode, and Andrew McGregor, *Robust lower bounds for communication and stream computation*, Theor. Comput. **12** (2016), no. 1, 1–35, Preliminary version in *Proc. 40th Annual ACM Symposium on the Theory of Computing*, pages 641–649, 2008.
- [57] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler, *Annotations in data streams*, ACM Trans. Alg. **11** (2014), no. 1, Article 7.
- [58] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian, *Verifiable stream computation and Arthur-Merlin communication*, Proc. 30th Annual IEEE Conference on Computational Complexity, 2015, pp. 217–243.
- [59] Amit Chakrabarti and Prantar Ghosh, *Streaming verification of graph computations via graph structure*, Proc. 33rd International Workshop on Randomization and Approximation Techniques in Computer Science, 2019, pp. 70:1–70:20.

- [60] Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova, *Vertex ordering problems in directed graph streams*, Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA), 2020, pp. 1786–1802.
- [61] Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl, *Adversarially Robust Coloring for Graph Streams*, 13th Innovations in Theoretical Computer Science Conference (ITCS 2022), vol. 215, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 37:1–37:23.
- [62] Amit Chakrabarti, Prantar Ghosh, and Justin Thaler, *Streaming Verification for Graph Problems: Optimal Tradeoffs and Nonlinear Sketches*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020), Leibniz International Proceedings in Informatics (LIPIcs), vol. 176, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 22:1–22:23.
- [63] Amit Chakrabarti and Sagar Kale, *Submodular maximization meets streaming: matchings, matroids, and more*, Math. Program. **154** (2015), no. 1–2, 225–247, Preliminary version in *Proc. 17th Conference on Integer Programming and Combinatorial Optimization*, pages 210–221, 2014.
- [64] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng, *The complexity of  $(\Delta+1)$  coloring in congested clique, massively parallel computation, and centralized local computation*, Proc. ACM Symposium on Principles of Distributed Computing, 2019, pp. 471–480.
- [65] Yi-Jun Chang, Wenzheng Li, and Seth Pettie, *An optimal distributed  $(\Delta+1)$ -coloring algorithm*, Proc. 50th Annual ACM Symposium on the Theory of Computing, 2018, pp. 445–456.

- [66] Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu, *Almost optimal super-constant-pass streaming lower bounds for reachability*, STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021, ACM, 2021, pp. 570–583.
- [67] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan, *One trillion edges: Graph processing at facebook-scale*, Proc. VLDB Endow. **8** (2015), no. 12, 1804–1815.
- [68] Fred C Chow and John L Hennessy, *The priority-based coloring approach to register allocation*, ACM Transactions on Programming Languages and Systems (TOPLAS) **12** (1990), no. 4, 501–536.
- [69] Don Coppersmith, Lisa Fleischer, and Atri Rudra, *Ordering by weighted number of wins gives a good ranking for weighted tournaments*, ACM Trans. Algorithms **6** (2010), no. 3, 55:1–55:13.
- [70] Graham Cormode, Jacques Dark, and Christian Konrad, *Independent sets in vertex-arrival streams*, Proc. 46th International Colloquium on Automata, Languages and Programming, 2019, pp. 45:1–45:14.
- [71] Graham Cormode, Michael Mitzenmacher, and Justin Thaler, *Streaming graph computations with a helpful advisor*, Algorithmica **65** (2013), no. 2, 409–442.
- [72] Graham Cormode and S. Muthukrishnan, *An improved data stream summary: the count-min sketch and its applications*, J. Alg. **55** (2005), no. 1, 58–75, Preliminary version in Proc. 6th Latin American Theoretical Informatics Symposium, pages 29–38, 2004.
- [73] Graham Cormode, Justin Thaler, and Ke Yi, *Verifying computations with streaming interactive proofs*, Proc. VLDB Endowment **5** (2011), no. 1, 25–36.

- 
- [74] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Proc. 6th Symposium on Operating System Design and Implementation, 2004, pp. 137–150.
- [75] Michael Elkin, *Distributed exact shortest paths in sublinear time*, Proc. 49th Annual ACM Symposium on the Theory of Computing, 2017, pp. 757–770.
- [76] P. Erdős and A. Hajnal, *On chromatic number of graphs and set-systems*, Acta Mathematica Academiae Scientiarum Hungarica **17** (1966), no. 1, 61–99.
- [77] Hossein Esfandiari, Silvio Lattanzi, and Vahab S. Mirrokni, *Parallel and streaming algorithms for  $k$ -core decomposition*, Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, 2018, pp. 1396–1405.
- [78] G. Even, J. (Seffi) Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica **20** (1998), no. 2, 151–174.
- [79] Martín Farach-Colton and Meng-Tsung Tsai, *Tight approximations of degeneracy in large graphs*, LATIN 2016: Theoretical Informatics, Springer Berlin Heidelberg, 2016, pp. 429–440.
- [80] Uriel Feige and Joe Kilian, *Zero knowledge and the chromatic number*, Annual IEEE Conference on Computational Complexity, 1996, p. 278.
- [81] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang, *On graph problems in a semi-streaming model*, Theor. Comput. Sci. **348** (2005), no. 2–3, 207–216, Preliminary version in *Proc. 31st International Colloquium on Automata, Languages and Programming*, pages 531–543, 2004.
- [82] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang, *Graph distances in the data-stream model*, SIAM J. Comput. **38** (2008),

- no. 6, 1709–1727, Preliminary version in *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, 2005.
- [83] Philippe Flajolet and G. Nigel Martin, *Probabilistic counting algorithms for data base applications*, J. Comput. Syst. Sci. **31** (1985), no. 2, 182–209.
- [84] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski, *Local conflict coloring*, Proc. 57th Annual IEEE Symposium on Foundations of Computer Science, 2016, pp. 625–634.
- [85] Eugene C. Freuder, *A sufficient condition for backtrack-free search*, J. ACM **29** (1982), no. 1, 24–32.
- [86] Sumit Ganguly and Graham Cormode, *On estimating frequency moments of data streams*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM 2007, Lecture Notes in Computer Science, vol. 4627, 2007, pp. 479–493.
- [87] Mohsen Ghaffari and Christiana Lymouri, *Simple and near-optimal distributed coloring for sparse graphs*, 31st International Symposium on Distributed Computing (DISC 2017), 2017, p. 20.
- [88] Mohsen Ghaffari and Ali Sayyadi, *Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication*, 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, 2019, pp. 142:1–142:14.
- [89] Prantar Ghosh, *New verification schemes for frequency-based functions on data streams*, Proc. 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2020, pp. 22:1–22:15.

- [90] Anna C. Gilbert and Piotr Indyk, *Sparse recovery using sparse matrices*, Proceedings of the IEEE **98** (2010), no. 6, 937–947.
- [91] Ashish Goel, Michael Kapralov, and Sanjeev Khanna, *On the communication and streaming complexity of maximum bipartite matching*, Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, 2012, pp. 468–485.
- [92] Oded Goldreich, *Introduction to property testing*, Cambridge University Press, 2017.
- [93] Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff, *Pseudo-Deterministic Streaming*, Proc. 20th Conference on Innovations in Theoretical Computer Science, vol. 151, 2020, pp. 79:1–79:25.
- [94] Tom Gur and Ran Raz, *Arthur–Merlin streaming complexity*, Proc. 40th International Colloquium on Automata, Languages and Programming, 2013, pp. 528–539.
- [95] Venkatesan Guruswami and Krzysztof Onak, *Superlinear lower bounds for multi-pass graph processing*, Algorithmica **76** (2016), no. 3, 654–683.
- [96] Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan, *Near-optimal distributed degree+1 coloring*, CoRR **abs/2112.00604** (2021), To appear in STOC 2022.
- [97] Moritz Hardt and David P. Woodruff, *How robust are linear sketches to adaptive inputs?*, Proc. 45th Annual ACM Symposium on the Theory of Computing, 2013, pp. 121–130.
- [98] David G Harris, Johannes Schneider, and Hsin-Hao Su, *Distributed  $(\Delta + 1)$ -coloring in sublogarithmic rounds*, Proc. 48th Annual ACM Symposium on the Theory of Computing, 2016, pp. 465–478.

- [99] Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu, *Greedy and local ratio algorithms in the mapreduce model*, Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018, 2018, pp. 43–52.
- [100] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer, *Adversarially robust streaming algorithms via differential privacy*, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.
- [101] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby, *A pseudo-random generator from any one-way function*, SIAM J. Comput. **28** (1999), no. 4, 1364–1396.
- [102] Monika Henzinger and Pan Peng, *Constant-time dynamic  $(\Delta+1)$ -coloring*, 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France, LIPIcs, vol. 154, 2020, pp. 53:1–53:18.
- [103] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan, *Computing on data streams*, External memory algorithms (1999), 107–118.
- [104] Zhiyi Huang, Sampath Kannan, and Sanjeev Khanna, *Algorithms for the generalized sorting problem*, IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011, 2011, pp. 738–747.
- [105] Piotr Indyk and David P. Woodruff, *Optimal approximations of the frequency moments of data streams*, Proc. 37th Annual ACM Symposium on the Theory of Computing, 2005, pp. 202–208.

- 
- [106] Madhav Jha, C. Seshadhri, and Ali Pinar, *A space efficient streaming algorithm for triangle counting using the birthday paradox*, Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 589–597.
- [107] Ce Jin, *Simulating random walks on graphs in the streaming model*, 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10–12, 2019, San Diego, California, USA, 2019, pp. 46:1–46:15.
- [108] Öjvind Johansson, *Simple distributed  $\Delta+1$ -coloring of graphs*, Information Processing Letters **70** (1999), no. 5, 229–232.
- [109] Hossein Jowhari, Mert Saglam, and Gábor Tardos, *Tight bounds for  $l_p$  samplers, finding duplicates in streams, and related problems*, Proc. 30th ACM Symposium on Principles of Database Systems, 2011, pp. 49–58.
- [110] A. B. Kahn, *Topological sorting of large networks*, Commun. ACM **5** (1962), no. 11, 558–562.
- [111] John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova, *The complexity of counting cycles in the adjacency list streaming model*, Proc. 38th ACM Symposium on Principles of Database Systems, 2019, pp. 119–133.
- [112] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun, *Counting arbitrary subgraphs in data streams*, Automata, Languages, and Programming, Springer Berlin Heidelberg, 2012, pp. 598–609.
- [113] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *On the exact space complexity of sketching and streaming small norms*, Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms, 2010, pp. 1161–1178.

- [114] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *An optimal algorithm for the distinct elements problem*, Proc. 29th ACM Symposium on Principles of Database Systems, 2010, pp. 41–52.
- [115] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer, *Separating adaptive streaming from oblivious streaming using the bounded storage model*, Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III, Lecture Notes in Computer Science, vol. 12827, Springer, 2021, pp. 94–121.
- [116] Michael Kapralov, *Better bounds for matchings in the streaming model*, Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms, 2013, pp. 1679–1697.
- [117] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon, *Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs*, Bioinform. **20** (2004), no. 11, 1746–1758.
- [118] Claire Kenyon-Mathieu and Warren Schudy, *How to rank with few errors: A PTAS for weighted feedback arc set on tournaments*, Electronic Colloquium on Computational Complexity (ECCC) **13** (2006), no. 144.
- [119] Shahbaz Khan and Shashank K. Mehta, *Depth first search in the semi-streaming model*, Proc. 36th International Colloquium on Automata, Languages and Programming, 2019, pp. 42:1–42:16.
- [120] Subhash Khot and Ashok Kumar Ponnuswami, *Better inapproximability results for maxclique, chromatic number and min-3lin-deletion*, International Colloquium on Automata, Languages and Programming, 2006, pp. 226–237.
- [121] L. Kirousis and D. Thilikos, *The linkage of a graph*, SIAM Journal on Computing **25** (1996), no. 3, 626–647.

- [122] Hartmut Klauck and Ved Prakash, *Streaming computations with a loquacious prover*, Proc. 4th Conference on Innovations in Theoretical Computer Science, 2013, pp. 305–320.
- [123] Hartmut Klauck and Ved Prakash, *An improved interactive streaming algorithm for the distinct elements problem*, Automata, Languages, and Programming - 41st International Colloquium (ICALP), LNCS, vol. 8572, 2014, pp. 919–930.
- [124] Kishore Kothapalli and Sriram Pemmaraju, *Distributed graph coloring in a few rounds*, Proc. 30th ACM Symposium on Principles of Distributed Computing, 2011, pp. 31–40.
- [125] Eyal Kushilevitz and Noam Nisan, *Communication complexity*, Cambridge University Press, Cambridge, 1997.
- [126] Frank Thomson Leighton, *A graph coloring algorithm for large scheduling problems*, Journal of research of the national bureau of standards **84** (1979), no. 6, 489–506.
- [127] Christoph Lenzen, *Optimal deterministic routing and sorting on the congested clique*, Proc. 32nd ACM Symposium on Principles of Distributed Computing, 2013, pp. 42–50.
- [128] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman, *The dynamics of viral marketing*, ACM Trans. Web **1** (2007), no. 1, 5–es.
- [129] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos, *Graphs over time: Density laws, shrinking diameters and possible explanations*, Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, 2005, p. 177–187.

- [130] Jure Leskovec and Andrej Krevl, *SNAP Datasets: Stanford large network dataset collection*, <http://snap.stanford.edu/data>, June 2014.
- [131] Feifei Li, Ke Yi, Marios Hadjieleftheriou, and George Kollios, *Proof-infused streams: Enabling authentication of sliding window queries on streams*, Proc. 33rd International Conference on Very Large Data Bases, 2007, pp. 147–158.
- [132] Vahid Lotfi and Sanjiv Sarin, *A graph coloring algorithm for large scale scheduling problems*, Computers & operations research **13** (1986), no. 1, 27–32.
- [133] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg, *Minimum-weight spanning tree construction in  $O(\log \log n)$  communication rounds*, SIAM J. Comput. **35** (2005), no. 1, 120–131.
- [134] Michael Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput. **15** (1986), no. 4, 1036–1053.
- [135] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan, *Algebraic methods for interactive proof systems*, J. ACM **39** (1992), no. 4, 859–868.
- [136] Andrew McGregor, *Finding graph matchings in data streams*, Proc. 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, 2005, pp. 170–181.
- [137] Andrew McGregor, *Graph stream algorithms: a survey*, ACM SIGMOD Record **43** (2014), no. 1, 9–20.
- [138] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu, *Densest subgraph in dynamic graph streams*, International Symposium on Mathematical Foundations of Computer Science, 2015, pp. 472–482.

- [139] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu, *Better algorithms for counting triangles in data streams*, Proc. 35th ACM Symposium on Principles of Database Systems, 2016, pp. 401–411.
- [140] Ilya Mironov, Moni Naor, and Gil Segev, *Sketching in adversarial environments*, SIAM J. Comput. **40** (2011), no. 6, 1845–1870.
- [141] Jayadev Misra and David Gries, *Finding repeated elements*, Sci. Comput. Program. **2** (1982), no. 2, 143–152.
- [142] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu, *Scalable large near-clique detection in large-scale networks via sampling*, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, 2015.
- [143] Farnaz Moradi, Tomas Olovsson, and Philippos Tsigas, *A local seed selection algorithm for overlapping community detection*, 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014), 2014, pp. 1–8.
- [144] Jelani Nelson, Huy L. Nguyễn, and David P. Woodruff, *On deterministic sketching and streaming for sparse recovery and norm estimation*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Lecture Notes in Computer Science, vol. 7408, 2012, pp. 627–638.
- [145] M. E. J. Newman, *The structure and function of complex networks*, SIAM Rev. **45** (2003), no. 2, 167–256.
- [146] Noam Nisan, *Pseudorandom generators for space-bounded computation*, Proc. 22nd Annual ACM Symposium on the Theory of Computing, 1990, pp. 204–212.

- 
- [147] Alessandro Panconesi and Aravind Srinivasan, *On the complexity of distributed network decomposition*, Journal of Algorithms **20** (1996), no. 2, 356–374.
- [148] Stavros Papadopoulos, Yin Yang, and Dimitris Papadias, *Cads: Continuous authentication on data streams*, Proc. 33rd International Conference on Very Large Data Bases, 2007, pp. 135–146.
- [149] Taehoon Park and Chae Y Lee, *Application of the graph coloring algorithm to the frequency assignment problem*, Journal of the Operations Research society of Japan **39** (1996), no. 2, 258–265.
- [150] Merav Parter,  *$(\Delta+1)$  coloring in the congested clique model*, Proc. 45th International Colloquium on Automata, Languages and Programming, 2018, pp. 160:1–160:14.
- [151] Merav Parter and Hsin-Hao Su, *Randomized  $(\Delta+1)$ -Coloring in  $O(\log^* \Delta)$  Congested Clique Rounds*, Proc. 32nd International Symposium on Distributed Computing, 2018, pp. 39:1–39:18.
- [152] Jaikumar Radhakrishnan, Saswata Shannigrahi, and Rakesh Venkat, *Hypergraph two-coloring in the streaming model*, arXiv preprint arXiv:1512.04188 (2015).
- [153] Alexander Razborov, *On the distributional complexity of disjointness*, Theor. Comput. Sci. **106** (1992), no. 2, 385–390, Preliminary version in *Proc. 17th International Colloquium on Automata, Languages and Programming*, pages 249–253, 1990.
- [154] Ryan A. Rossi and Nesreen K. Ahmed, *The network data repository with interactive graph analytics and visualization*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
- [155] Václav Rozhon and Mohsen Ghaffari, *Polylogarithmic-time deterministic network decomposition and distributed derandomization*, Proc. 52nd Annual ACM Symposium on the Theory of Computing, 2020.

- 
- [156] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy, *Estimating pagerank on graph streams*, J. ACM **58** (2011), no. 3, 13.
- [157] Johannes Schneider and Roger Wattenhofer, *A new technique for distributed symmetry breaking*, Proc. 29th ACM Symposium on Principles of Distributed Computing, 2010, pp. 257–266.
- [158] Adi Shamir, *IP = PSPACE*, J. ACM **39** (1992), no. 4, 869–877.
- [159] Uri Stemmer, *Separating adaptive streaming from oblivious streaming*, Lecture at STOC 2021 Workshop: Robust Streaming, Sketching and Sampling, available online at <https://www.youtube.com/watch?v=svgv-xw9DZc&t=7679s>, 2021, Based on joint work with Haim Kaplan, Yishay Mansour, and Kobbi Nissim.
- [160] Michael Stiebitz and Bjarne Toft, *A Brooks type theorem for the maximum local edge connectivity*, Electronic Journal of Combinatorics **25** (2016), P1.50.
- [161] George Szekeres and Herbert S. Wilf, *An inequality for the chromatic number of a graph*, Journal of Combinatorial Theory **4** (1968), no. 1, 1 – 3.
- [162] Robert Endre Tarjan, *Edge-disjoint spanning trees and depth-first search*, Acta Informatica **6** (1976), no. 2, 171–185.
- [163] Justin Thaler, *Data stream verification*, Encyclopedia of Algorithms, Springer New York, 2016, pp. 494–499.
- [164] Justin Thaler, *Semi-streaming algorithms for annotated graph streams*, Proc. 43rd International Colloquium on Automata, Languages and Programming, 2016, pp. 59:1–59:14.
- [165] Simon Thevenin, Nicolas Zufferey, and Jean-Yves Potvin, *Graph multi-coloring for a job scheduling application*, Discrete Applied Mathematics **234** (2018), 218–235.

- [166] Peter A. Tucker, David Maier, Lois M. L. Delcambre, Tim Sheard, Jennifer Widom, and Mark P. Jones, *Punctuated data streams*, 2005.
- [167] Stanley Wasserman and Katherine Faust, *Social network analysis: Methods and applications*, vol. 8, Cambridge university press, 1994.
- [168] David P. Woodruff, *Optimal space lower bounds for all frequency moments*, Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 167–175.
- [169] David P. Woodruff and Samson Zhou, *Tight bounds for adversarially robust streams and sliding windows via difference estimators*, 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021), Denver, CO, USA, February 7-10, 2022, 2021, pp. 1183–1196.
- [170] Andrew C. Yao, *Some complexity questions related to distributive computing*, Proc. 11th Annual ACM Symposium on the Theory of Computing, 1979, pp. 209–213.
- [171] Amir Yehudayoff, *Pointer chasing via triangular discrimination*, Tech. Report TR16-151, ECCC, 2016.
- [172] Ke Yi, Feifei Li, Marios Hadjieleftheriou, George Kollios, and Divesh Srivastava, *Randomized synopses for query assurance on data streams*, Proc. 24th International Conference on Data Engineering, 2008, pp. 416–425.
- [173] David Zuckerman, *Linear degree extractors and the inapproximability of max clique and chromatic number*, Proc. 38th Annual ACM Symposium on the Theory of Computing, 2006, pp. 681–690.