Optimal Graph Streaming Algorithms and Further Advances in Modern Models of Computation

by

Vihan Shah

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Vihan Shah 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Slobodan Mitrović, Assistant Professor Department of Computer Science University of California, Davis
Supervisor(s):	Sepehr Assadi, Associate Professor Department of Computer Science University of Waterloo
Internal Member:	Eric Blais, Associate Professor Department of Computer Science University of Waterloo
Internal-External Member:	Jochen Koenemann, Professor Department of Combinatorics & Optimization University of Waterloo
Other Member(s):	Elena Grigorescu, Professor Department of Computer Science University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

All the results of this thesis are based on papers that I have coauthored, as well as unpublished work of mine. The details are as follows:

- The content of Chapter 3 is based on a recent work with Matthew Ding, Alexandro Garces, Jason Li, Honghao Lin, Jelani Nelson, and David P. Woodruff [DGL+25] published in ITCS 2025.

- The content of Chapter 4 is based on a paper co-authored with Sepehr Assadi [AS23a], published at SOSA 2023.

- The content of Chapter 5 is based on two papers: one co-authored with Sepehr Assadi [AS22], published at ITCS 2022, and a student-only paper with Kheeran K. Naidu [NS22], published at APPROX 2022.

- The content of Chapter 6 is based on a paper co-authored with Sepehr Assadi and Chen Wang [ASW23], published at NeurIPS 2023.

- The content of Chapter 7 is based on a paper co-authored with Sepehr Assadi [AS25], published at PODS 2025.

- The content of Chapter 8 is based on a paper co-authored with Vladimir Braverman, Prathamesh Dharangutte, and Chen Wang [BDSW24], published at APPROX 2024.

- The content of Chapter 9 is based on a paper co-authored with Prantar Ghosh [GS24], published at ITCS 2024.

- The content of Chapter 10 is based on a solo-authored manuscript that is currently under submission [Sha25].

Abstract

The rise of large-scale datasets across domains such as social networks, biological systems, and the web has made it increasingly important to understand how core graph problems can be solved under tight resource constraints. As these datasets grow, traditional algorithms that assume random access to the input become increasingly infeasible. This thesis explores how to process massive graphs efficiently under modern models of computation that address these limitations. The primary focus is on the streaming model, and this thesis also explores other modern models, including sublinear-time, fully dynamic, and oracle-based models.

The first part of the thesis develops space-optimal algorithms for fundamental graph problems in the streaming setting. We study approximate minimum cut, k-vertex connectivity, maximum matching, minimum vertex cover, and correlation clustering across different streaming models—including insertion-only, dynamic, and random-order streams. By establishing tight upper and lower bounds—often matching up to constant or polylogarithmic factors—these results resolve several open questions in the streaming literature and characterize entire space-approximation trade-offs for some of these problems.

The second part of the thesis expands the exploration to other modern models of computation, including sublinear-time algorithms, fully dynamic algorithms, and oracle-based models, such as learning-augmented algorithms and streaming verification. We begin by studying the 4-cycle counting problem in the fully dynamic model and show an improvement over the previous best algorithm, demonstrating that the previously assumed natural bound was not tight. In the sublinear setting, we examine the problem of estimating matching size and prove strong lower bounds against non-adaptive algorithms. For the maximum independent set problem in the learning-augmented model, we develop a new algorithm that achieves a significantly improved approximation factor in polynomial time. Lastly, we explore the streaming verification model, focusing primarily on connectivity problems. Together, these contributions deepen our understanding of the fundamental limits and possibilities of algorithm design for massive data under constrained computational resources.

Acknowledgements

I am deeply grateful to my advisor, Sepehr Assadi, for his unwavering support and steady mentorship throughout my PhD journey. His hands-on, intellectually demanding yet encouraging style created an environment where I was constantly challenged to grow. He was patient with my early struggles but never shied away from being honest — even brutally so when I needed a wake-up call. Those moments, though difficult, became turning points that shaped my work ethic and approach to research. Working with Sepehr has been transformative; he fundamentally shaped the way I think about problems, always pushing for clarity, depth, and precision. His high standards inspired me to raise my own, and his feedback on presentations and writing has been invaluable. I've always appreciated how generous he is with his time, and his commitment to academic integrity has made a deep impression on me. And of course, I'll always smile at how "good" has unintentionally become a part of my own vocabulary — a small reflection of his influence. I am truly thankful to have had the opportunity to learn from him.

My heartfelt thanks go to Rajiv Gandhi, who first introduced me to the world of algorithms and sparked my interest in theoretical computer science. Attending the Program in Algorithmic and Combinatorial Thinking (PACT) in 2019, which he leads with such passion and care, was a pivotal moment in my academic journey. His guidance not only provided the foundation I needed, but also made it possible for me to transfer to Rutgers and discover the joy of research. Through his mentorship and encouragement, I found the confidence and motivation to pursue a PhD. His support has profoundly shaped my academic path, and I remain deeply appreciative of the inspiration he has given me.

The members of my advisor's group — my academic siblings — have also been an integral part of this journey. Chen Wang has been both a collaborator and mentor; his incredible work ethic has always inspired me. Janani Sundaresan has been endlessly generous with her time, helping me work through difficult proofs, and her determination and brilliance have left a lasting impression. Parth Mittal has not only been a thoughtful sounding board for ideas, but also a great friend — from research discussions to climbing walls, he's taught me how to think more clearly. I'm also thankful to Chaitanya Nalam and Hoai-An Nguyen, whose curiosity and enthusiasm have contributed to the energy of our group. Finally, it has been a pleasure to welcome Helia Yazdanyar as the newest member of the group, and I look forward to seeing all that she will bring. I'm grateful to each of them for making our group so special.

The professors and postdocs at Rutgers played a formative role in my academic development. In particular, I'm thankful to Prantar Ghosh for his mentorship, collaboration,

and generous support. He has been a trusted mentor and collaborator, and his guidance — especially during the postdoc application process — has meant a great deal to me. I also learned a lot from others in the theory group during my time there, including Michael Saks, Jie Gao, Aaron Bernstein, Karthik C.S., Zihan Tan, Nicole Wein, Roei Tell, and Ariel Schvartzman. Their lectures, discussions, and reading groups played a meaningful role in shaping my interests and understanding of the field.

My time at the University of Waterloo was equally shaped by the support of professors and postdocs who guided me during the latter part of my PhD. I'd like to thank Eric Blais, Rafael Oliveira, Lap Chi Lau, Elena Grigorescu, Robert Andrews, and Ingrid Vukusic for their insights and encouragement. I am especially thankful to Robert Andrews for his thoughtful conversations and guidance that helped me refine and articulate my research vision. I'm also especially grateful to Eric Blais for taking the time to go through my solo-authored paper with me — offering detailed feedback, helping verify the proof, and even catching a subtle bug I had overlooked. Their mentorship contributed greatly to both my academic development and confidence as a researcher.

Over the course of my PhD, I've had the opportunity to collaborate with many brilliant professors, postdocs, and students. I'm grateful to Vladimir Braverman, Prathamesh Dharangutte, Matthew Ding, Alexandro Garces, Prantar Ghosh, Nirmit Joshi, Jason Li, Honghao Lin, Kheeran K. Naidu, Jelani Nelson, Shreyas Pai, Milind Prabhu, Chen Wang, and David P. Woodruff. These collaborations have been deeply enriching, and I've learned a great deal from each one.

I would like to thank the members of my dissertation committee for their time and willingness to serve in this important role. I am grateful to Slobodan Mitrović for agreeing to be the external examiner. I also appreciate the support and participation of Eric Blais and Elena Grigorescu. I am especially thankful to Jochen Koenemann for serving as the internal-external member.

Several professors and postdocs outside Rutgers and Waterloo have also played a meaningful role in my journey. I thank Christian Konrad for his generous help during my postdoc applications. I'm grateful to Sanjeev Khanna for teaching a guest course at Waterloo that offered valuable insights. I've also had the pleasure of collaborating with Ali Vakilian and Sepideh Mahabadi, and I'm thankful for the opportunity to learn from them. Finally, I thank David Tench for introducing me to the practical side of streaming algorithms.

The student communities in the Theoretical Computer Science (TCS) groups at both Rutgers and Waterloo have been a constant source of support, energy, and camaraderie. At Rutgers, I'm especially grateful to Vikrant Ashvinkumar, Chengyuan Deng, Prathamesh Dharangutte, Hanna Komlos, Vishwas Bhargava, Aditi Dudeja, Surya Teja Gavva, Rash-

mika Goswami, Zach Langley, Adarsh Srinivasan, Guido Tagliavini, and Harsha Tirumala for making my time there more engaging and enjoyable through countless conversations, shared challenges, and friendships. At Waterloo, I'm thankful to Avantika Agarwal, Vahid R. Asadi, Omkar Baraskar, Abhibhav Garg, Prashant Gokhale, Arkaprava Choudhury, Mohammad Hossein Ebtehaj, Renato Ferreira, Sriram Gopalakrishnan, Cameron Seth, Ajay Singh, John Smith, Abhiroop Sanyal, Stephanie Maaz, Argyris Mouzakis, Rian Neogi, Matthew Regehr, Remy El Sabeh, Dante Tjowasi, Kam Chuen (Alex) Tung, Dinis Vitorino, and Robert Wang for creating a welcoming and intellectually stimulating environment. I feel lucky to have been part of such thoughtful and supportive communities.

Helpful discussions with Zachary Langley, Michael Saks, Christian Konrad, Vikrant Ashvinkumar, Abhibhav Garg, Prashant Gokhale, Zhiang Wu, and Samson Zhou contributed to various aspects of my papers, and I'm grateful for their insights.

I feel incredibly fortunate to have had the support of friends who stood by me through different phases of this journey. Among the people who meant the most were Rushit Patel, Nirmit Mehta, Dhruv Parekh, Neil Shah, Parva Mujpura, Vidhi Parekh, Malvika Patrachari, Riya Doshi, Tanvi Mankame, and Sharvari Mankame — my weekly support system, who kept me grounded and sane through it all. Rushit, in particular, reminded me to enjoy life and not let work take over completely. Upon arriving in the US, Adney Cardoza, Srushti Nandu, Aayushi Purohit, and Vaibhavi Hansrajani helped me find my footing and make the transition so much easier. Several friendships from undergrad — Jashay Lilani, Chayan Rellan, Kiran Teja Tummuri, Vikram Sagi, Dhruv Srivastava, Krishi Kattunga, Aniketh Reddy, Naval Pattar, Kartheek Mylavarapu, Shreyas Rajesh, Tuhin Bagi, and Nagesh Eranki — remained a constant source of support, with many of them having moved to the US as well. Having Prem Chandran and Pavithra Gopalakishnan in Canada brought warmth and companionship that I'm deeply thankful for. India still feels close thanks to Dhruv Prakash, Shobhit Mishra, Mahima Vadgama, Karishma Wadkar, Nihal Jain, Vedant Lakhani, and Yash Bhawnani, whose continued presence and encouragement have meant a lot. A special thanks to Dhruv Prakash for always hyping me up and making me believe in myself whenever I needed it. Their friendship, in all its forms, has been a constant source of strength and joy.

None of this would have been possible without the unwavering love and support of my family. To my mother and father — thank you for your endless sacrifices, for always believing in me, and for being my biggest pillars of strength through every high and low. Your encouragement gave me the courage to dream big, and your faith kept me going when things got tough. To my sister, thank you for being my constant cheerleader and my closest friend — your love and humor have meant the world to me. I am forever grateful to have you all by my side.

Dedication

This thesis is dedicated to my parents, Nandita and Jitesh Shah, and my sister, Dhriti Shah, whose constant encouragement and quiet sacrifices made this journey possible.

Table of Contents

Examining Committee Membership ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vi

Dedication ix

List of Figures xvii

List of Tables xx

1 Introduction 1

1.1 Graph Streaming 2

1.2 Overview of Previous Work 4

1.3 Our Contributions: Optimal Graph Streaming 7

1.4 Other Modern Models of Computation 11

1.4.1 Sublinear-time Algorithms 11

1.4.2 Dynamic Algorithms 12

1.4.3 Oracle Models . . . 13
1.5 Our Contributions: Modern Models . . . 15
1.6 A Reader's Manual . . . 20

2 Preliminaries **22**
2.1 Notation . . . 22
2.2 Streaming and Linear Sketches . . . 23
2.3 Standard Tools . . . 25
2.4 Probabilistic Tools . . . 25
2.5 Techniques . . . 27

I Part I: Optimal Graph Streaming Algorithms **30**

3 Streaming Minimum Cut **31**
3.1 Introduction . . . 31
3.1.1 Our Techniques . . . 35
3.2 Preliminaries . . . 37
3.3 Streaming For-each Sparsifier . . . 38
3.4 Streaming Algorithm for Approximate Minimum Cut . . . 41
3.5 Approximate Minimum Cut Lower Bounds . . . 43
3.6 Lower Bound: Approximate Minimum Cut . . . 45
3.6.1 Warm up: Exact Minimum Cut . . . 45
3.6.2 Approximate Minimum Cut . . . 47
3.7 Exact Minimum Cut in Random-Order Streams . . . 48

4 Streaming Vertex Connectivity **54**
4.1 Introduction . . . 54
4.2 A Certificate of Vertex Connectivity . . . 58

4.2.1 Proof of Lemma 4.2.4 . . . 62
4.2.2 Proof of Lemma 4.2.5 . . . 65
4.3 The Dynamic Streaming Algorithm . . . 66
4.4 The Lower Bound . . . 68

5 Maximum Matching in Dynamic Streams 71
5.1 Introduction . . . 71
5.2 New Sketching Toolkit . . . 75
5.2.1 Neighborhood-Edge Sampler . . . 75
5.2.2 Sparse-Neighborhood Recovery . . . 78
5.2.3 Neighborhood-Size Tester . . . 88
5.3 Main Result and Setup . . . 91
5.4 Main Algorithm: Handling the Sparsify-Case . . . 94
5.4.1 The Sketching Matrix . . . 95
5.4.2 The Recovery Algorithm . . . 106
5.5 Match-Or-Sparsify Lemma . . . 111
5.5.1 First Batch: Proof of Lemma 5.5.3 . . . 115
5.5.2 Second Batch: Proof of Lemma 5.5.4 . . . 117

6 Estimating Correlation Clustering Cost in Streaming 127
6.1 Introduction . . . 127
6.1.1 Our Contributions . . . 129
6.1.2 Our Techniques . . . 131
6.2 Preliminaries . . . 132
6.2.1 Notation . . . 132
6.2.2 Problem Definition . . . 133
6.2.3 The Streaming Model . . . 134
6.2.4 Standard Technical Tools . . . 135

6.3 Purely additive approximation . . . 135
6.4 A Lower Bound for $O(n^{2-\varepsilon})$ Additive error . . . 139
6.4.1 A construction of correlation clustering structural . . . 140
6.4.2 Multi-copy correlation clustering structure . . . 145
6.4.3 Cost Testing Lower Bound – A Variate of the Hard Instance . . . 147
6.5 A Lower Bound for $O(n)$ Additive error . . . 151
6.5.1 Gap Cycle Counting with Odd Cycles . . . 152
6.5.2 The lower bound . . . 156

II Part II: Modern Models of Computation 159

7 Fully Dynamic 4-Cycle Counting 160
7.1 Introduction . . . 160
7.2 Preliminaries . . . 167
7.3 Warm-up: When A and C are Fixed . . . 169
7.3.1 Setup . . . 169
7.3.2 Data Structures . . . 171
7.3.3 Queries: . . . 174
7.3.4 Constraints: . . . 175
7.4 Setup for the Main Algorithm . . . 176
7.5 Main Algorithm . . . 178
7.5.1 Phases . . . 179
7.5.2 Data Structures . . . 180
7.5.3 Queries . . . 182
7.5.4 Pseudocode . . . 185
7.6 Getting Rid of Assumption 3: Tiny Vertices . . . 187
7.6.1 Data Structure for Tiny Vertices . . . 188

7.6.2 Queries for Tiny Vertices . . . 190
7.6.3 Paths Through Tiny Vertices . . . 190
7.7 Getting Rid of Assumption 4: Vertices Moving Between Classes . . . 192
7.7.1 Setup . . . 192
7.7.2 Proof of Lemma 7.7.1 . . . 195
7.8 Extending the Algorithm to 4-Cycles in General Graphs . . . 201

8 Oracle models: Learning-Augmented Maximum Independent Set 203
8.1 Introduction . . . 203
8.1.1 Our models and contributions . . . 205
8.1.2 Technical overview . . . 206
8.2 An Algorithm in the Persistent Noise Setting . . . 208
8.3 An Algorithm in the Non-persistent Noise Setting . . . 211

9 Oracle Models: Streaming Verification 220
9.1 Introduction . . . 220
9.1.1 Motivation and Context . . . 220
9.1.2 Our Results and Contributions . . . 224
9.1.3 Related Work . . . 227
9.1.4 Technical Overview . . . 229
9.1.5 Paper Organization . . . 238
9.2 Models, Notation, and Terminology . . . 238
9.2.1 Models . . . 238
9.2.2 Notation and Terminology . . . 244
9.3 Preliminaries . . . 245
9.4 OMA Lower Bound for Equals-Index and its Implications . . . 246
9.4.1 The Equals-Index Problem . . . 247
9.4.2 Implications of the Equals-Index Lower Bounds . . . 249

9.5 Classical Streaming Algorithms under SGT streams . . . 258
9.5.1 Basic Tools: Non-Zero Detection and ℓ_0-Sampling for Large Frequencies . . . 258
9.5.2 Edge Connectivity . . . 265
9.6 Annotated Streaming Schemes for Dynamic Graph Streams . . . 273
9.6.1 The Layering Lemma . . . 273
9.6.2 Vertex Connectivity Schemes . . . 279
9.6.3 Edge Connectivity Schemes . . . 282
9.7 Annotated Streaming Schemes for SGT Streams . . . 283

10 Sublinear-Time Non-Adaptive Matching Size **286**
10.1 Introduction . . . 286
10.1.1 Prior Work . . . 287
10.1.2 Our Contributions . . . 288
10.1.3 Technical Overview . . . 289
10.1.4 Related Work . . . 291
10.2 Preliminaries . . . 292
10.3 The Lower Bound . . . 293
10.3.1 The Hard Instance Construction . . . 294
10.3.2 Properties of the Hard Instance . . . 297
10.3.3 Limitations of the Algorithm . . . 299
10.4 The Coupling Argument . . . 307
10.4.1 Coupling the Core . . . 312
10.4.2 Fixing the Permutation and the Star Centers . . . 314
10.4.3 Fixing edges to D and Adjacency Lists of Non-Petals . . . 316
10.4.4 Fixing Final Blocks . . . 320
10.5 The Algorithm . . . 327

References **331**

APPENDICES **380**

A Missing details from Chapter 4 **381**

A.1 An Insertion-Only Streaming Algorithm 381

A.2 Mader's Theorem 384

B Missing details from Chapter 5 **386**

B.1 Sparse-Neighborhood Recovery via Exhaustive-Search 386

C Missing details from Chapter 7 **389**

C.1 A Simple Algorithm for Counting 4-Cycles 389

C.2 Verifying Constraints 390

D Missing details from Chapter 8 **393**

D.1 A Simple Algorithm for an $O(\log n)$-approximation in the Non-persistent Noise Setting 393

D.2 A Lower Bound for the Non-persistent Noise Setting 394

D.2.1 A lower bound for obtaining the entire maximum independent set . 395

D.2.2 A lower bound for obtaining an $O(1)$-approximation of the maximum independent set 397

List of Figures

3.1 Construction of graph H^+ for exact minimum cut from [Zel11]. Red nodes are the ones that are added in addition to the original graph. 46

3.2 Construction of graph $H^+(\varepsilon)$ for approximate minimum cut. Red nodes are the ones that are added in addition to the original graph. G_i represents the graph containing (potential) edge ab, and G_0 and G_1 represent the remaining graphs. The double arrow edges represent bicliques between graph $G_0 \in G_S$ and S, and graph $G_1 \in G_T$ and T. 48

4.1 An illustration of the partition (S, X, T) of vertices of G and H. There are no edges between S and T in H, while G has at least one edge $e = (s, t)$ between S and T, to ensure its k-vertex-connectivity as $|X| < k$. 61

4.2 An illustration of the s-t paths $P_1(X), P_2(X), \ldots, P_k(X)$. Each $P_j(X)$ consists of an edge a_j from s, a path B_j until the last edge c_j to t. 63

4.3 An illustration of a good graph G_{i^*} wherein both vertices s and t are sampled and all the vertices in set X are not. Thus, none of the s-t paths, except for the edge e, exist in G_{i^*} since they all pass through X. Therefore, the spanning forest T_{i^*} necessarily contains the edge $e = (s, t)$. 66

5.1 This figure shows S and its neighborhood $N(S)$ which intersects with T. T has size at most a and $N(S) - T$ has size at most b. Also, every vertex v in $N(S) - T$ has at most c neighbors in S. 79

5.2 Illustration of edges that satisfy the first condition, but not other conditions of weakly-represented edges. 98

5.3 This figure shows a strongly-represented edge (in blue). There is exactly one edge e of $\overline{G}_{\text{easy}}$ between V_i and V_j both of which are non-expanding and clean. Thus, e is strongly-represented. 98

5.4 Illustration of two groups V_i and V_j that pass the M_{easy}-test and Expanding-test. We have that $NR(V_i)$ has only one element u in V_j and $NR(V_j)$ has only one element v in V_i. This means that (u, v) must be an edge in G. . 107

5.5 Illustration of random variables X_v. 122

6.1 The $\mathsf{BHH}_{m,t}$ instance to Gap Cycle Counting with odd cycles. In the instance of the figure, Alice has $x_3 = 1$ (for matching j). Therefore, we have $Mx = 1$ and the graph is one 14-length cycle. One can verify that if one more set of 'cross edges' is added, the graph goes to two separate cycles. 155

7.1 Binary Relations A, B. Every element of the join $(A \bowtie B)$ corresponds to a 2 -paths in the layered graph between L_1 and L_3. Thus, the size of the join $(A \bowtie B)$ is the number of 2-paths in the layered graph. 168

7.2 This figure shows a query (u, v) where $u \in H$ and $v \in M$. We iterate over the neighbors of v and then use the wedges to u stored in the data structure $A^{H*} \cdot B_{<i}$. The neighbors of v are shown using red edges, the edges in $B_{<i}$ are in gray and the wedges to u are shown in blue. 174

7.3 This figure shows a query (u, v) where $u \in H$ and $v \in M$. We iterate over the neighbors of v (shown in red) and then use the wedges through sparse vertices to u stored in the data structure $A^{*S}B^{S*}$ (shown in blue). To get the DD paths we use the wedges through dense vertices to u stored in the data structure $A^{HD}B^{DD}$ (shown in green). To get the DS paths we need to iterate over the neighbors of u in L_2^D and use the wedges through sparse vertices to v stored in the data structure $B^{*S}C^{S*}$ (not shown here). 183

9.1 Construction for XOR-CONN$_n$ 254

9.3 An illustration of the cut S in G and H. After X is deleted, there are no edges between S and T in H, while G has at least one edge $e = (s, t)$ between S and T, to ensure its k-edge-connectivity as $|X| < k$. 267

9.4 An illustration of the k edge-disjoint s-t paths $P_1(X), P_2(X), \ldots, P_k(X)$. Note that the paths can share vertices but are edge-disjoint. 269

9.5 An illustration of a good graph G_{i^*} wherein the edge $e = (s, t)$ is sampled and all the edges in set X are not. Thus, none of the s-t paths, except for the edge e, exist in G_{i^*}. Therefore, the spanning forest T_{i^*} necessarily contains the edge $e = (s, t)$. 271

10.1 Hard Distribution 290

10.2 Illustration of a Yes and a No instance. Blue edges denote identity matchings: a matching of size 8 in the Yes case, and two matchings of size 4 each in the No case. Pink and Green edges form matchings that are structurally identical across the instances but connect to different vertex sets on the left. Orange squiggly lines represent edges from D^L to $A^R \cup B^R$ (same for D^R), existing with probability roughly 1/2. 296

10.3 Showing the block-compliant matching $\mathcal{M}$. 317

List of Tables

1.1 A list of some of our optimal graph streaming results in insertion-only, dynamic, and random-order streams. 7

3.1 Minimum Cut Space Complexity in the Single-Pass Insertion-Only Streaming Setting 35

7.1 Data Structures used in the Algorithm where A and C are fixed. 171

7.2 Data Structures used in the Main Algorithm. 180

7.3 Data Structures for Tiny Vertices. 188

10.1 Notation for Graph Components and Parameters. 293

10.2 Notation for the Distributions, Randomness, and Observed Graph. 293

10.3 Notation for Section 10.3.3. 300

10.4 Notation for Section 10.4. 308

10.5 Notation introduced in Section 10.4.1. 313

10.6 Notation for Sections 10.4.2 and 10.4.3. 316

10.7 Notation for Section 10.4.4. 321

Chapter 1

Introduction

What are efficient algorithms? If we go back 50 years, the answer would have been algorithms that run in polynomial time. Edmonds, in his seminal paper [Edm65], gave the first polynomial-time algorithm for non-bipartite maximum matching—a breakthrough that influenced the formal development of computational complexity theory. His work helped shape the definition of complexity classes such as P and NP, laying the foundation for how we understand algorithmic efficiency.

Fast forward a few decades, and the bar has moved. As datasets began to grow dramatically, research shifted toward designing *almost-linear time algorithms*, which became the new gold standard of efficiency. Many fundamental problems now admit such algorithms, including edge coloring [ABB+05], spectral sparsification [ST11], minimum cut [Kar00], maximum flow and bipartite matching [CKL+22, VDBCK+23, AKL+22], vertex connectivity [LNP+21], and negative weight shortest paths [BNWN25] among many others. This gave rise to a rich and active research area known as fast graph algorithms.

These advances have primarily focused on graph problems, which lie at the core of many foundational algorithmic questions. Graphs are ubiquitous, and countless real-world scenarios can be modeled using them. They provide a structured way to analyze problems and have applications in areas such as network design, biological systems, social networks, and electrical circuits. This thesis focuses on graph problems, which have long been central to theoretical computer science.

While classical algorithms made significant progress under the assumption that the entire input fits in memory and is readily accessible, this assumption no longer holds in many modern applications. Today's graphs are often so large that even linear-time algorithms become infeasible. In practice, graphs may be stored on disk or streamed

from external sources. In such cases, random access to the input becomes prohibitively expensive, and the assumptions of classical computational models break down.

Based on a large user study [SMS+20], modern graph workloads often involve *billions of edges*—20 of 89 users reported graphs with over 10^9 edges, and some with up to 10^{11}. These graphs span domains ranging from social and biological networks to enterprise data. Users commonly solve problems such as connectivity and matching, with *scalability* to billion-edge graphs being the most pressing challenge.

This trend is reflected in large-scale datasets across diverse domains. Earlier examples already demonstrated massive scale: Microsoft's Live Messenger connected 180 million users via 1.3 billion links [LH08]; telecom networks log billions of monthly calls [SMS+08]; and protein–protein interaction networks aggregate up to 200 million interactions [RTC+14]. Social networks also reached significant sizes early on—Facebook and Twitter have each published social graphs with hundreds of millions of users and billions of edges [UKBM11, KLPM10]. Since then, graphs have grown even larger. The Hyperlink Web graph contains 3.5 billion vertices and 128 billion edges [DBS21, MVL+15], and models of the human brain involve around 10^{10} nodes [McG14]. Facebook's social graph is now estimated to contain over a trillion edges [CEK+15].

To address these challenges, we turn to modern models of computation that more accurately capture the resource constraints of large-scale data processing. In many settings, random access is prohibitively expensive, whereas sequential scans are much more efficient [HRR98, GKS05, Vit01]. This naturally suggests processing the graph incrementally, handling pieces of it as they arrive. This leads to the *streaming model*, which is the primary focus of this thesis. In this model, the input arrives as a sequence of updates and must be processed using memory that is sublinear in the input size. Streaming is particularly well-suited for massive graphs, where even storing the full input in memory is infeasible.

In addition to streaming, other modern models also provide tools for large-scale graph processing. The *sublinear model* allows algorithms to read only a small portion of the input and return approximate solutions, while the *dynamic model* maintains solutions as the graph evolves through insertions and deletions. While the primary focus of this thesis is on streaming algorithms, we also explore sublinear and dynamic models as further examples of modern approaches for processing massive graphs.

1.1 Graph Streaming

The streaming model was first formalized by Alon, Matias, and Szegedy [AMS96]. In this model, the data arrives in an arbitrary order and the algorithm has to process it on the fly.

Once the stream terminates, the algorithm has to solve the problem using the information it has stored. One way to solve a problem in this setting is to store the entire stream and then solve the problem offline. As we saw earlier, the data is often too large to store in its entirety, so the goal is to minimize the amount of space used. In the context of graph streaming, the m edges of the graph arrive in a stream, and the vertices are fixed from 1 to n. Thus, the goal is to minimize the space used, ideally achieving $o(n^2)$ space.

Streaming algorithms can be used anywhere where data is supposed to be processed on the fly, such as in network routers, financial markets, and incoming satellite data. Streaming can also be used in scenarios where the data is fully stored but is difficult to access efficiently, for instance data on a hard drive that exceeds the available RAM size. In this case, accessing the hard drive sequentially in a streaming fashion is far more efficient than random access.

Semi-Streaming. Many classical graph problems require $\Omega(n)$ bits of space to output their solutions. Motivated by this, [FKM+04] in their seminal work nearly two decades ago, introduced the *semi-streaming* model, which is defined by its memory budget of $\widetilde{O}(n)$ bits of space. This space is nearly proportional to the number of vertices, but is usually much smaller than the number of edges. This motivates the natural question: What graph problems admit semi-streaming algorithms?

Dynamic Streams. In some applications, the stream must account for changes in the dataset, such as in social media graphs, where edges can be added or removed. This suggests the need for *dynamic streams*, which allow for both insertions and deletions of edges, with the same edge potentially being inserted and deleted multiple times. The problem has to be solved on the final graph, where an edge is present if it is inserted more times than it is deleted.

At first glance, it is unclear whether meaningful computation is even possible in this setting. Even seemingly simple tasks—like outputting a single edge—pose challenges: if we store an edge it may be deleted later. In contrast, this task is trivial in insertion-only streams, where one can simply return the first edge. In a surprising result, Ahn and Guha [AGM12a] showed that this problem can be solved in $\text{polylog}\,(n)$ space in dynamic streams. Randomization is crucial: any deterministic algorithm requires $\Omega(n^2)$ space in the worst case to output even a single edge.

Building on this foundational tool, a large body of work has emerged on dynamic streaming algorithms for a variety of graph problems. This led to a fascinating phenomenon: many insertion-only streaming algorithms were adapted to the dynamic setting with only a $\text{polylog}\,(n)$ factor overhead in space. This naturally raises the question:

Which insertion-only streaming algorithms can be adapted to dynamic streams with only a polylog n factor increase in space? We will make progress towards these questions in this thesis.

1.2 Overview of Previous Work

In this section, we first review some of the foundational results in the streaming setting and then narrow in on some important problems that we will tackle in this thesis.

Traditionally, research in the streaming model has focused on problems such as frequency moments [AMS96, KNW10b], quantile estimation [MRL99, GK01, AJPS23], and heavy hitters [CM05], among others (see also [Mut05]).

Over time, the focus of streaming algorithms expanded to include graph problems. Henzinger et al. [HRR98] were among the earliest to explore graph problems in the streaming setting. As discussed earlier, any graph problem can be trivially solved in $O(n^2)$ space by storing the entire stream and running an offline algorithm. The challenge, then, is to design algorithms that use significantly less space. To formalize this objective, Feigenbaum et al. [FKM+04] introduced the *semi-streaming model*, which restricts space usage to $\widetilde{O}(n)$ bits.

This choice is well-motivated. Many fundamental graph problems—such as connected components, minimum spanning tree, and minimum cut—require $\Omega(n)$ bits even to output their solutions. Furthermore, Feigenbaum et al. [FKM+05] showed that even decision versions of problems like bipartiteness, connectivity, and minimum spanning tree require $\Omega(n)$ bits of space. Along these lines, Sun and Woodruff [SW15] established $\Omega(n \log n)$ bit lower bounds for a wide range of graph problems in the streaming model, including Eulerian testing, cycle-freeness, planarity, and $O(1)$-approximate minimum cut—reinforcing the idea that $o(n)$ space is fundamentally infeasible for many graph problems. Since many of these problems admit algorithms within $\widetilde{O}(n)$ space, this regime has emerged as the natural *sweet spot* for the design of graph streaming algorithms.

The foundational papers [FKM+04, FKM+05] also studied key problems such as maximum matching, graph distances, and spanners. These papers popularized the study of graph problems in the streaming setting. Since then, a wide range of graph problems have been studied in the streaming model, including subgraph counting [BYKS02, BFL+06, KMSS12, BOV13, MVV16, KMPV19, MV20, Vor20], spanners [EZ04, bas, Bas08, Elk11], clique and independent set [HHLS10, HSSW12], densest subgraph [BKV12], maximum cut [KKS14b, KKSV17], and vertex coloring [ACK19b, AKM22, AY25], among many others.

Many fundamental graph problems like matching, vertex cover, minimum cut, vertex connectivity, and correlation clustering require $\Omega(n^2)$ bits of space to solve exactly. This motivates the study of approximation algorithms for them within the semi-streaming model. We now review the state of the art for each of these problems, highlighting key results in both insertion-only and dynamic streaming settings.

Minimum Cut. Minimum cut and cut sparsification are central problems in the graph streaming literature. Computing the exact minimum cut requires $\Omega(n^2)$ space [Zel11]. To overcome this, Ahn et al. [AG09] gave an algorithm for $(1+\varepsilon)$-approximate cut sparsifiers using $\widetilde{O}(n/\varepsilon^2)$ space. Since such sparsifiers preserve all cuts up to a $(1+\varepsilon)$ factor, they also suffice to approximate the minimum cut within the same approximation. [AG09] further established a lower bound of $\Omega(n\log(1/\varepsilon))$ bits for approximating the minimum cut. More recently, [ACK+16, CKST19] proved a nearly matching lower bound of $\widetilde{\Omega}(n/\varepsilon^2)$ for cut sparsifiers, showing that the upper bound is tight up to polylogarithmic factors. While the complexity of sparsifier construction is now well understood, there remains a significant gap between the known upper and lower bounds for approximating the minimum cut.

These results apply to insertion-only streams. To extend these guarantees to dynamic settings, the breakthrough works of Ahn, Guha, and McGregor [AGM12a, AGM12b] used linear sketching to recover an edge from a dynamic stream using only $\widetilde{O}(1)$ space. Building on this primitive, they developed dynamic streaming algorithms for several problems, including connectivity, bipartiteness, minimum spanning tree, k-edge-connectivity, and cut sparsifiers. Their bounds for cut sparsification in dynamic streams matched the insertion-only setting—$\widetilde{O}(n/\varepsilon^2)$—up to polylogarithmic factors. The dynamic streaming complexities of connectivity and bipartiteness also matched their insertion-only complexities, revealing a surprising phenomenon: many graph problems exhibit similar space complexity in both insertion-only and dynamic streams, up to polylogarithmic factors.

k-Vertex Connectivity. Their bounds for k-edge connectivity were $\widetilde{O}(kn)$, matching the best known results in insertion-only streams up to polylogarithmic factors. In a follow-up work, Guha, McGregor, and Tench [GMT15] studied a different version of the problem: k-vertex connectivity. They obtained a bound of $\widetilde{O}(k^2n)$, whereas the corresponding insertion-only bound was $\widetilde{O}(kn)$. This left open whether the additional factor of k is unavoidable in dynamic streams.

Maximum Matching. One of the most extensively studied graph problems across many models is the maximum matching problem. A large body of work has focused on multi-pass algorithms [FKM+04, McG05, KMM12, Kap13, GO16, KT17, AR20, KN21, FMU21, AS23b, FS24]. In contrast, progress in the single-pass setting has been limited. It is well known

that an exact maximum matching requires $\Omega(n^2)$ space, while a simple greedy algorithm yields a 2-approximation in $\widetilde{O}(n)$ space. A recent breakthrough by Assadi et al. [ABKL23] provided the first non-trivial improvement over this baseline in a single pass. Still, the landscape for single-pass matching in insertion-only streams remains poorly understood.

In comparison, the complexity of maximum matching in dynamic streams is much better characterized. [Kon15, AKLY16, CCE+16, DK20] showed that the space complexity of computing an α-approximate matching is $\widetilde{\Theta}(n^2/\alpha^3)$. Whether the polylogarithmic factors in this bound are necessary remained an open question. Notably, any constant-factor approximation in dynamic streams requires $\Omega(n^2)$ space, whereas a 2-approximation is achievable in insertion-only streams using just $\widetilde{O}(n)$ space. This gap demonstrates that the phenomenon of similar complexities between insertion-only and dynamic streams does not extend to all problems. Vertex cover is another such example: the same works also show that computing an α-approximate vertex cover in dynamic streams has complexity $\widetilde{\Theta}(n^2/\alpha^2)$ space. It remains a central open question to characterize which graph problems exhibit similar space complexity in insertion-only and dynamic streams, up to polylogarithmic factors.

Correlation Clustering. Finally, we look at the correlation clustering problem in the streaming model—a fundamental optimization problem that lies at the intersection of machine learning and theoretical computer science. This problem has been studied in many models like sublinear time [BGK13, AW22], Massively Parallel Computation [CLM+21, BCMT22], and the dynamic model [BDP+25, CALMP24, CCAL+25a]. [AW22] gave the first $O(1)$-approximation algorithm in semi-streaming and [BCMT23, MC23, CKL+24] improved the approximation constant to 3. It was then shown that it is possible to get a $(1+\varepsilon)$-approximation in $\widetilde{O}(n/\varepsilon^2)$ space [BCMT23, AKP25] resolving the problem in semi-streaming. All these results require $\Omega(n)$ space, which is reasonable since outputting the full solution itself takes $\Omega(n)$ bits. In contrast, if the goal is to just estimate the cost of the clustering, it is natural to ask whether $o(n)$ space could suffice. This left open the question of whether such an algorithm exists.

These results show that while some graph problems have similar space complexity in insertion-only and dynamic streams, others exhibit stark separations. Although prior work has developed approximation algorithms for many of these problems, the exact space-approximation tradeoffs are still not fully understood. In the next section, we address these open questions by developing optimal algorithms and proving matching lower bounds.

1.3 Our Contributions: Optimal Graph Streaming

We now present our main contributions in the graph streaming setting. We go over what our main results are in each chapter.

A central theme in the first part of this thesis is the pursuit of *optimal streaming algorithms* for fundamental graph problems. We study problems such as minimum cut, k-vertex connectivity, maximum matching, vertex cover, and correlation clustering. For each of these, we resolve open questions by establishing tight upper and lower bounds—matching up to polylogarithmic or even constant factors. These results settle the streaming space complexity of the corresponding problems and, in the case of matching and vertex cover, eliminate polylogarithmic overhead to achieve asymptotically optimal bounds.

Another central theme in the first part of this thesis is understanding the full *space-approximation tradeoff*, rather than focusing solely on bounds in semi-streaming. By exploring the entire tradeoff, we gain a more nuanced view of the problem and uncover structural insights that can potentially lead to improved techniques in other computational models. Our results for matching, vertex cover, and minimum cut provide a nearly complete characterization of this tradeoff. We extend these ideas of achieving tradeoffs to several chapters in the second part of the thesis as well.

Problem	**Model**	**Upper Bound**	**Lower Bound**
$(1+\varepsilon)$-Minimum Cut	Insertion	$\widetilde{O}(n/\varepsilon)$ (Chapter 3)	$\Omega(n/\varepsilon)$ (Chapter 3)
$(1+\varepsilon)$-Minimum Cut	Dynamic	$\widetilde{O}(n/\varepsilon^2)$ [AGM12b]	$\Omega(n/\varepsilon)$ (Chapter 3)
Exact Minimum Cut	Random	$\widetilde{O}(n)$ (Chapter 3)	$\Omega(n)$ [CCM16]
k-Edge Connectivity	Dynamic	$\widetilde{O}(kn)$ [AGM12a], Chapter 9	$\Omega(kn)$ [SW15]
k-Vertex Connectivity	Dynamic	$\widetilde{O}(kn)$ (Chapter 4)	$\Omega(kn)$ [SW15], Chapter 4
α-approx Matching	Dynamic	$O(n^2/\alpha^3)$ (Chapter 5)	$\Omega(n^2/\alpha^3)$ [DK20]
α-approx Vertex Cover	Dynamic	$O(n^2/\alpha^2)$ (Chapter 5)	$\Omega(n^2/\alpha^2)$ [DK20]

Table 1.1: A list of some of our optimal graph streaming results in insertion-only, dynamic, and random-order streams.

Chapter 3: Streaming Minimum Cut. We begin by considering the minimum cut problem in the insertion-only streaming setting. To solve the problem exactly, any algorithm needs $\Omega(n^2)$ space [Zel11], which essentially corresponds to storing the entire stream.

However, if we only need a $(1+\varepsilon)$-approximation to the minimum cut, then the problem can be solved in $\widetilde{O}(n/\varepsilon^2)$ space [AG09]. This approach approximately preserves not only the minimum cut but also all cuts up to a factor of $(1+\varepsilon)$, namely finding a cut-sparsifier of the input graph, a much stronger property. There is a lower bound showing that any data structure for a cut-sparsifier needs $\Omega(n \log n/\varepsilon^2)$ bits of space [ACK+16,CKST19][1]. But the best streaming lower bound for minimum cut was only $\Omega(n \log 1/\varepsilon)$ bits of space [AG09]. Thus, resolving the dependence on ε was an open problem.

In this thesis, we resolve this open problem by giving a nearly *space-optimal* algorithm that uses $\widetilde{O}(n/\varepsilon)$ space, showing that the extra factor of $1/\varepsilon$ is not necessary for the minimum cut problem. We also give a matching lower bound of $\Omega(n/\varepsilon)$ bits resolving the complexity of this problem up to polylogarithmic factors. Cut sparsifiers are a natural tool to solve approximate minimum cut, but any approach to get a better upper bound could not go through them. Thus, the main tool we use is called a for-each sparsifier which is similar to a cut sparsifier, but it only preserves any fixed cut with high probability instead of preserving all cuts with high probability. This problem has recently been studied in the classical setting [ACK+16,CGP+18]. We use this result and combine it with the merge and reduce framework in streaming to solve the problem using $\widetilde{O}(n/\varepsilon)$ space. This approach crucially relies on randomization, and we show that it is not possible to derandomize it by giving an $\Omega(n/\varepsilon^2)$ bit lower bound for deterministic algorithms (this is optimal because the upper bound of [AG09] can be made deterministic).

These results give a near optimal understanding of $(1+\varepsilon)$-approximate minimum cut. However, is there a way to relax the model so that we can solve for exact minimum cut in semi-streaming space? [RSW18, AD21] addressed this by allowing two passes over the stream and could solve the problem exactly in $\widetilde{O}(n)$ space. We relax the problem by assuming that the edges of the graph arrive in a random order instead of an adversarial order. This assumption defines the random-order streaming model, under which we show that it is indeed possible to determine the exact minimum cut in $\widetilde{O}(n)$ space. It was already known that the decision version of this problem needed $\Omega(n)$ bits of space [CCM16], so we resolve the complexity of this problem up to polylogarithmic factors as well.

The content of this chapter is based on a recent work with Matthew Ding, Alexandro Garces, Jason Li, Honghao Lin, Jelani Nelson, and David P. Woodruff [DGL+25] published in ITCS 2025.

Streaming Chapter 4: k-Vertex Connectivity. We now turn our attention to the k-vertex-connectivity problem, which asks the question "Is the vertex connectivity of the

[1]This implies that any streaming algorithm that approximately preserves all cuts also needs $\Omega(n \log n/\varepsilon^2)$ bits of space.

graph less than k or at least k?" In the insertion-only streaming model, an upper bound of $\widetilde{O}(kn)$ bits was known by the sparsification techniques of [CKT93, EGIN97] and [SW15] gave a nearly matching lower bound of $\Omega(kn)$ bits. In contrast, as discussed earlier, for dynamic streams, the best known upper bound was $\widetilde{O}(k^2n)$ bits [GMT15]. It remained an open question whether the extra factor of k was necessary for dynamic streams.

In this thesis, we give a nearly *space-optimal* algorithm for this problem in $\widetilde{O}(kn)$ space by giving a model-independent approach for computing a k-vertex-connectivity certificate of size $\widetilde{O}(kn)$. A certificate of k-vertex-connectivity is a subgraph of the original graph that is k vertex connected if and only if the original graph is. Consider the following process to construct the certificate. We sample every vertex with probability $1/k$ and then pick an arbitrary maximal spanning forest in the graph induced by the sampled vertices. The certificate then is just a union of $O(k \log n)$ independent instances of this process. We also give an efficient implementation of this certificate in dynamic streams in $\widetilde{O}(kn)$ space using existing results. This result shows that the dynamic streaming complexity matches the insertion-only complexity up to polylogarithmic factors, closing a significant gap in our understanding of dynamic versus insertion-only streams.

Finally, we extend our algorithm to the multi-pass setting and show that we can solve the problem in p-passes using $\widetilde{O}(kn/p)$ bits of space. We also provide a nearly matching lower bound of $\Omega(kn/p)$ bits for any p-pass streaming algorithm that solves the problem.

The content of this chapter is based on a paper co-authored with Sepehr Assadi [AS23a], published at SOSA 2023.

Chapter 5: Streaming Matching and Vertex Cover. We now consider one of the most well-studied problems in the streaming setting: maximum matching, along with its closely related problem, minimum vertex cover. As we saw earlier, many graph problems exhibit similar complexities in both the insertion-only and dynamic streaming models up to polylogarithmic factors. Maximum matching and minimum vertex cover stand out as notable exceptions. These problems were addressed in dynamic streams in a series of works [Kon15, AKLY16, CCE+16, DK20] which showed that any constant approximation needs $\Omega(n^2)$ space for both maximum matching and minimum vertex cover. These works resolved the bounds up to polylogarithmic factors, providing space bounds of $\widetilde{\Theta}(n^2/\alpha^3)$ and $\widetilde{\Theta}(n^2/\alpha^2)$ for an α-approximation for maximum matching and minimum vertex cover respectively. It was believed that these additional polylogarithmic factors were necessary because there were many streaming lower bounds for graph problems where polylogarithmic factors appeared in lower bounds [SW15, NY19] and no upper bounds had been achieved without them. For instance, [NY19] proved that any dynamic streaming algorithm for

connectivity requires $\Omega(n \log^3 n)$ bits of space which matches the algorithm of [AGM12a] up to constant factors.

In this thesis, we surprisingly show that the space upper bound for maximum matching does not need the polylogarithmic factors by designing an algorithm that uses $O(n^2/\alpha^3)$ *bits* of space (for any $\alpha \ll n^{1/2}$).[2] The main tool we developed for this problem, called "Match or Sparsify" either gives us a large matching or reduces the problem to finding a large matching in a sparse graph. We also extend these ideas to the minimum vertex cover problem and design an algorithm that uses $O(n^2/\alpha^2)$ *bits* of space for any $\alpha \ll n$. These results resolve the space complexity of matching and vertex cover in dynamic streams up to *constant* factors, showing that it is indeed possible for natural problems to achieve asymptotically optimal space complexity without additional polylogarithmic factors in the space bound. An interesting aspect of the vertex cover result is that when α is very large, the space of the algorithm is much smaller than the vertex cover it outputs. This is possible because the solutions we output are highly structured.

The content of this chapter is based on two papers: one co-authored with Sepehr Assadi [AS22], published at ITCS 2022, and a student-only paper with Kheeran K. Naidu [NS22], published at APPROX 2022.

Chapter 6: Streaming Correlation Clustering Value. We now investigate the *correlation clustering* problem in the streaming model. Prior work on streaming correlation clustering has primarily focused on the *semi-streaming* setting. After a series of works, it was recently shown that a $(1+\varepsilon)$-approximation can be achieved in $\widetilde{O}(n/\varepsilon^2)$ space [BCMT23, AKP25], effectively resolving the problem within the semi-streaming regime. But what if we wish to go below semi-streaming space? In this work, we investigate streaming algorithms that operate with significantly more stringent memory constraints—using only polylog (n) bits of space. This space is not enough to output the entire clustering of the graph. Instead, we focus on estimating the *correlation clusterability* of the input graph; that is, approximating the cost of the optimal correlation clustering solution without explicitly constructing the clustering itself.

We present three novel algorithms that, using only polylog (n) space, are able to estimate the optimal correlation clustering cost up to a constant multiplicative factor plus an additional additive error. Our main algorithm achieves a 1-multiplicative approximation with an additive error of $o(n^2)$. We also prove an almost matching lower bound, showing that if the additive error is required to be $n^{2-\delta}$ for some $\delta > 0$, then any algorithm

[2] The space of the algorithm $O(n^2/\alpha^3)$ has to be at least as large as the space required to output an α-approximate matching which is at least n/α, implying $\alpha \leqslant \sqrt{n}$.

achieving such a purely additive approximation must use more than polylog (n) space. Finally, we present another lower bound that justifies the mix of multiplicative and additive approximation in our remaining algorithms.

The content of this chapter is based on a paper co-authored with Sepehr Assadi and Chen Wang [ASW23], published at NeurIPS 2023.

1.4 Other Modern Models of Computation

1.4.1 Sublinear-time Algorithms

We now move beyond streaming and explore other modern models of computation that are used in this thesis. At the beginning of this chapter, we noted that modern datasets are often so large that even linear-time algorithms are no longer practical. While streaming algorithms address this challenge by limiting space usage, *sublinear-time* algorithms approach it from a different angle—by limiting the time spent on the input. In this model, the algorithm is only allowed to inspect a small portion of the input, often through a limited number of queries. Instead of computing exact outputs, these algorithms typically settle for approximate answers with high probability. The goal is to solve decision problems or estimate quantities without reading the entire input, thereby achieving runtimes that are significantly smaller than linear.

The area of sublinear algorithms originated in the 1990s with the introduction of property testing by [BLR90], and was later formalized by [RS96, GGR98] (see [Gol17, BY22, Rub90, Fis04a, Gol99, APR13, KR03] for additional resources). Early problems studied in this framework included testing graph properties such as bipartiteness and k-colorability [GGR98], estimating numerical parameters like the average degree [Fei04b, GR08] and the cost of a minimum spanning tree [CRT05], as well as testing function properties like linearity [Rub90], and distributional properties such as uniformity and independence [BFF+01].

The field has since grown significantly, with many graph problems studied in recent years, including subgraph counting [ELRS17, AN22, AKK19] vertex coloring [ACK19b, AY25], correlation clustering [AW22], and maximal independent set [AS19]. Maximum matching is one of the most extensively studied problems in the sublinear-time setting as well. While it is not possible to output an approximate matching in sublinear time, the size of a maximum matching can be estimated within sublinear time. A notable result by Behnezhad [Beh22] established a 2-approximation in $\widetilde{O}(n)$ time. On the lower bound side, [BRR23] showed that $\Omega(n^{1.2-o(1)})$ queries are necessary to achieve better than a 2/3-approximation.

Despite these significant advancements, our understanding of the landscape remains incomplete, with many fundamental questions still unanswered and notable gaps between upper and lower bounds. While these results have significantly advanced our understanding of sublinear algorithms for matching size, they all rely on adaptive querying strategies, where queries are chosen based on answers to previous queries. A natural question is whether similar approximations can be achieved in a fully non-adaptive setting, where all queries must be fixed in advance.

1.4.2 Dynamic Algorithms

The dynamic model is another modern model of computation, closely related to the streaming model. Here, the input arrives as a sequence of updates—such as insertions or deletions—but unlike streaming, the primary goal is to minimize the *time* required to process each update, rather than space. A key distinction is that the problem must be solved *after every update*, making the *update-time* crucial. The standard complexity measure in this setting is the *worst-case update time*, which denotes the maximum time required to maintain a valid solution after any single update. A relaxed measure, also commonly studied, is the *amortized update time*, which averages the cost over a sequence of updates. The most general and widely studied variant is the *fully dynamic model*, which allows both insertions and deletions of elements. This thesis will focus on this fully dynamic setting.

The dynamic model was introduced in the early 1980s, initially in the context of dynamic trees and minimum spanning forests [ST81, Fre85]. Eppstein et al. [EGIN97], with their technique of sparsification, improved update times for a range of graph problems, including connectivity, minimum spanning forests, edge and vertex connectivity, and shortest paths. Since then, the model has been widely applied to a variety of graph problems. Some examples include dynamic connectivity [KKM13, GHN+23], shortest paths [GWN20], and maximum matching, for which there is a large and growing body of work [OR10, GP13, BGS15, BS15, BS16, BK22, Beh23, ABKL23, BKS23a, Liu24, BG24, BKSW24, AS25]. For a broader overview of recent developments in fully dynamic graph algorithms, see the survey by Hanauer et al. [HHS22].

Subgraph counting is a popular graph problem and has been studied in various settings [AYZ97, JWZ24, VWW25, NRR14, KMPV19, MV20, Vor20, ELRS17, AKK18] but there was only a little work done on it in the dynamic model (see [HHS22]). The problem of 3-cycle counting in the dynamic model is well-studied and has upper and lower bounds of $\Theta(m^{1/2})$ for the worst-case update time [KNN+20, HKNS15]. However, even the bounds for 4-cycle counting in the dynamic model are not well understood. There is an upper bound of $O(m^{2/3})$ [HHH22] for the worst-case update time, but the conditional lower bound is

the same bound of $\Omega(m^{1/2})$. The algorithm of [HHH22] is a natural generalization of the approach for counting triangles in [KNN+20], so it seems natural that $O(m^{2/3})$ might be the correct answer for the update time for counting 4-cycles but no such formal lower bound was proved.

1.4.3 Oracle Models

Learning Augmented Algorithms. Another modern model of computation that has gained popularity in recent years is that of learning-augmented algorithms. In this framework, a classical algorithm is given access to an oracle that provides noisy predictions about the solution. This setup allows the algorithm to potentially achieve improved complexity when the predictions are accurate, while still maintaining worst-case guarantees when the predictions are poor. One can think of the oracle as a modern machine-learning model—typically highly accurate on predictive tasks, but prone to random and inconsistent errors. Learning-augmented algorithms thus offer a principled way to design and analyze algorithms beyond the worst-case paradigm, often leading to significant practical improvements.

The motivation behind learning-augmented algorithms arises from the growing gap between theoretical guarantees and practical performance in algorithm design. While worst-case analysis is crucial for ensuring reliability, real-world data is often highly structured—structure that classical algorithms typically fail to exploit. On the other hand, machine learning methods can effectively leverage such structure but generally lack rigorous performance guarantees. Learning-augmented algorithms aim to combine the best of both approaches: they integrate machine-learned predictions into classical algorithms, using the adaptability of learning while maintaining worst-case bounds. These algorithms are designed to perform very well when predictions are accurate, yet degrade gracefully when they are not—achieving a balance between practical efficiency and theoretical robustness.

Learning-augmented algorithms have attracted considerable attention in recent years [PSK18, HIKV19, LMRX21, CHW22, SW23, BGTZ23, BCGL23, ACN+23, HLS+23, BFNP24, SE24]. Several *graph problems*—including cycle counting [CEI+22], graph coloring [ABM23], shortest paths [LSV23], and maximum cut [CdG+24]—have also been explored in this framework.

In recent years, several works such as [ASS20, PSYC21, BQS+23] have leveraged machine learning techniques to design algorithms for the *Maximum Independent Set* (MIS) problem. However, this problem has not yet been formally studied within the framework of learning-augmented algorithms. Finding a maximum independent set is a classical NP-hard problem [Kar10], and moreover, it is NP-hard to approximate the size of the MIS to

within a factor of $n^{1-\delta}$ for any constant $\delta > 0$ [Hås96, Zuc06]. Thus, it remains an open problem whether better approximations can be achieved in the presence of predictions.

Streaming Verification. Another oracle model considered in this thesis is the *streaming verification* model also known as *annotated streaming*, which is a version of the streaming setting where a space-limited streaming algorithm—referred to as the verifier—receives help from an all-powerful but untrusted *prover* with unlimited space. After observing the data stream, the verifier is given a message by the prover, who aims to convince the verifier of the correctness of a claimed solution. The goal is for the verifier to use this additional information to validate the result while operating with significantly less space than would be required to compute it alone.

The motivation for studying streaming verification comes from the limitations of traditional streaming algorithms, which often require storing almost the entire graph—making them impractical for very large datasets. Streaming verification is useful in scenarios like cloud computing, where a resource-limited client has to process a massive data stream. Instead of solving the problem alone, the client offloads the computation to a powerful server. However, since the server might be untrustworthy or make mistakes, the client requires not just an answer but also a short proof that the answer is correct. With only a limited view of the stream, the verifier can then efficiently check the proof, ensuring the solution is correct without having to store the entire input.

In many cases, streaming verification enables exponentially lower space usage compared to classical streaming algorithms, often reducing the space requirement from $\Omega(n)$ to polylogarithmic in n with the help of an appropriate proof. However, this space reduction typically comes at the cost of a large proof size.

Chakrabarti et al. [CCMT14] showed a fundamental tradeoff: if h denotes the size of the proof and v the verifier's space, then any (h, v)-scheme must satisfy $h \cdot v = \Omega(s)$, where s is the classical (unannotated) streaming complexity of the problem. The main complexity measure in this setting is the sum $h + v$, but obtaining meaningful tradeoffs between h and v is a central challenge. Ideally, one would hope for a smooth tradeoff, where h and v can be chosen freely as long as $h \cdot v = \widetilde{O}(s)$. However, such tradeoffs are not always achievable.

Thaler [Tha16b] showed that any (h, v)-scheme for connectivity must satisfy $(h+n) \cdot v \geqslant n^2$. This implies that although connectivity can be solved without help in $\widetilde{O}(n)$ space, obtaining a verifier that uses $o(n)$ space requires help of size $\omega(n)$. Such results motivate the study of *non-trivial complexity*, which considers schemes where the verifier uses space $v = o(s)$. In the case of connectivity, the non-trivial complexity is only slightly larger than n, but it remains an open question whether certain problems require much higher non-trivial complexity.

Lower bounds in streaming verification are usually proved via reductions from the *Online Merlin-Arthur* (OMA) communication model [CCMT14]. This model is a variant of the one-way communication setting, where Alice and Bob each hold part of the input and only Alice sends a message to Bob. In the OMA model, a third party—Merlin—knows both inputs and sends a help message to Bob, who then uses both messages to compute the output. Merlin is untrusted, so the protocol must be sound even if his message is incorrect. Annotated streaming schemes can be viewed as to OMA protocols, just as classical streaming algorithms can be viewed as communication protocols. As a result, lower bounds in the OMA model translate directly into lower bounds for streaming verification. This framework was used by Thaler [Tha16b] to establish the lower bound for connectivity. However, there has been no prior work on k-edge connectivity or k-vertex connectivity in this model.

Comparison of Oracle Models. Although learning-augmented algorithms and streaming verification arise from different motivations—machine-learned predictions in one case and untrusted provers in the other—they share several similarities. Both models enhance a resource-constrained algorithm with access to an external, potentially unreliable oracle. In the learning-augmented setting, the oracle provides potentially inaccurate predictions, while in streaming verification, the prover may be malicious and attempt to convince the algorithm of a claimed result.

Despite this unreliability, both frameworks have strong guarantees: learning-augmented algorithms are robust to prediction errors, and streaming verifiers are designed to catch incorrect claims with high probability. These models aim to go beyond traditional lower bounds by augmenting the computational model. Learning-augmented algorithms may beat worst-case bounds when predictions are good, and streaming verification algorithms can verify solutions to problems in space far below what's needed to compute them from scratch.

Learning-augmented algorithms have also been studied in the streaming setting [JLL+20, CEI+22, BV24, DPV25], where predictions arrive alongside the stream as annotations. This setting closely resembles the annotated streaming model, with the key difference being that the annotations in streaming verification are structured as proofs rather than predictions.

1.5 Our Contributions: Modern Models

We now present our main contributions in other modern models of computation. We go over what our main results are in each chapter.

In the second part of the thesis, we explore four other modern models of computation

and solve problems in each of them. We begin with maintaining the count of 4-cycles in a graph within the dynamic model. We then examine oracle-based models. The first of these is the learning-augmented model, where we study the maximum independent set problem. The second is the streaming verification model, in which we investigate k-vertex and k-edge connectivity. Finally, we revisit the maximum matching problem in the sublinear-time model, focusing on estimating the matching size using non-adaptive queries.

Chapter 7: Dynamic 4-Cycle Counting. Our focus now shifts to other modern models of computation, beginning with subgraph counting in the dynamic setting. Subgraph counting is a fundamental problem in graph theory with wide-ranging applications, including social network analysis, telecommunication networks, and computational biology. It also plays a central role in database theory [NRR14, NPRR18, KNN+20], where the problem of counting the number of elements in binary joins over binary relations is equivalent to the problem of subgraph counting in *layered graphs*.

We are generally interested in maintaining the number of k-cycles. The problem of 3-cycle counting in the dynamic model is well-studied and has an upper bound of $O(m^{1/2})$ for the worst-case update time [KNN+20]. There is also a conditional lower bound of $\Omega(m^{1/2-\varepsilon})$ for the update time [HKNS15] under the Online Matrix-Vector conjecture, for any constant $\varepsilon > 0$.

However, even the bounds for 4-cycle counting in the dynamic model are not well understood. There is an upper bound of $O(m^{2/3})$ [HHH22] for the worst-case update time, but the conditional lower bound is the same bound of $\Omega(m^{1/2-\varepsilon})$ for any constant $\varepsilon > 0$ [HKNS15, HHH22]. The algorithm of [HHH22] is a natural generalization of the approach for counting triangles in [KNN+20], so it seems natural that $O(m^{2/3})$ might be the correct answer for the update time for counting 4-cycles. They also studied the problem of counting 4-cliques showing that the folklore upper bound of $O(m)$ for the update time is tight under the static combinatorial 4-clique conjecture by giving a lower bound of $\Omega(m^{1-\gamma})$ for any $\gamma > 0$. Thus, it seems natural that $O(m^{2/3})$ might be the correct answer for the complexity of the update time for counting 4-cycles.

In this thesis, we show this is not the case by giving an algorithm for maintaining the number of 4-cycles in fully dynamic graphs in worst-case update time $O(m^{2/3-\delta})$ for some constant $\delta > 0$. This shows that $O(m^{2/3})$ is not the correct answer for the complexity of the update time. This algorithm crucially uses fast matrix multiplication and surprisingly needs $\omega < 2.5$.[3] Using the current best value of the square matrix multiplication exponent $\omega = 2.3713391$, we get $\delta = 0.009811$, and if we assume the best possible exponent, i.e.

[3]The algorithm gives no improvement if we use an algorithm with matrix multiplication exponent $\geqslant 2.5$ like Strassen's algorithm. This is not common for algorithms based on fast matrix multiplication.

$\omega = 2$, then we get $\delta = 1/24$. We also show that the problem of counting 4-cycles is equivalent in layered graphs and general graphs.

The content of this chapter is based on a paper co-authored with Sepehr Assadi [AS25], published at PODS 2025.

Chapter 8: Learning Augmented Maximum Independent Set. Our focus now shifts to the *Maximum Independent Set* (MIS) problem, a classical NP-hard problem [Kar10]. Moreover, the seminal works of [Hås96, Zuc06] establish that approximating the size of the MIS to within a factor of $n^{1-\delta}$, for any constant $\delta > 0$, is also NP-hard. In contrast, simply outputting any single vertex yields a trivial n-approximation. [BH92] provided the first non-trivial approximation algorithm with an $O(n/\log^2 n)$ guarantee, which was subsequently improved by [Fei04a]. These results highlight the inherent difficulty of the MIS problem in general graphs, motivating a substantial body of research focused on developing approximation algorithms in special settings.

In recent years, there has been growing interest in *learning-augmented algorithms*—a paradigm that goes beyond worst-case analysis by leveraging predictions from machine-learned oracles. This approach has proven effective in overcoming classical algorithmic barriers in a variety of settings. This naturally raises the question: can such predictions help improve approximations for the MIS problem? In this work, we show that it is indeed possible to break classical approximation barriers with access to a learned oracle. Specifically, we consider an oracle that answers *vertex membership queries* for a fixed but unknown MIS, returning the correct answer with probability $1/2 + \varepsilon$. In the first setting we study, each vertex can be queried exactly once, and the oracle returns whether that vertex belongs to the fixed MIS with the aforementioned probability. Under this model, we design an algorithm that achieves an $\widetilde{O}(\sqrt{\Delta}/\varepsilon)$-approximation in $O(m)$ time, where Δ is the maximum degree of the graph. This gives an $\widetilde{O}(\sqrt{n})$-approximation in the worst case, when ε is a constant, representing a substantial improvement over the classical $n/\text{polylog}\,(n)$-approximation.

We then explore a relaxed setting in which the oracle can be queried multiple times per vertex, with each response being correct independently with probability $1/2+\varepsilon$, using fresh randomness each time. While it is straightforward to see that with $O(n \log n)$ total queries, one could recover the exact MIS with high probability, we consider a more constrained regime where the total number of oracle queries is limited to $O(n)$—the same as in the previous setting. A simple strategy in this regime is to randomly sample a $1/\log n$ fraction of the vertices and query each of them $\log n$ times, as in the previous approach, yielding a $\log n$-approximation. However, we go further and design a more sophisticated algorithm that achieves an $O(1)$-approximation using only $O(n)$ *total* queries when ε is constant,

with an overall runtime of $\widetilde{O}(m)$.

The content of this chapter is based on a paper co-authored with Vladimir Braverman, Prathamesh Dharangutte, and Chen Wang [BDSW24], published at APPROX 2024.

Chapter 9: Streaming Verification. We next consider another oracle model, namely the streaming verification model, and study fundamental graph problems such as k-edge connectivity and k-vertex connectivity. Additionally, we look at the *Merlin-Arthur* (MA) communication model, which serves as a key tool for proving lower bounds in the streaming verification setting. Our results establish new lower bounds in both models.

The MA communication model is an enhancement of the classical communication model, where in addition to the usual players Alice and Bob, there is an all-powerful but untrusted player Merlin who knows their inputs and tries to convince them about the output. Most functions have MA protocols with total communication significantly smaller than what would be needed without Merlin. We focus on the online MA (OMA) model, which is the MA analogue of one-way communication, and introduce the notion of *non-trivial-OMA* complexity of a function. This is the minimum total communication needed by any non-trivial OMA protocol computing that function, where a trivial OMA protocol is one where Alice sends Bob roughly as many bits as she would have sent without Merlin.

We prove a lower bound on the non-trivial-OMA complexity of a natural function *Equals-Index* (basically the well-known Index problem on large domains) and identify it as a canonical problem for proving strong lower bounds on this complexity: reductions from it (i) reproduce and/or improve upon the lower bounds for all functions that were previously known to have large non-trivial-OMA complexity, (ii) exhibit the first explicit functions whose non-trivial-OMA complexity is superlinear, and even exponential, in their classical one-way complexity, and (iii) show functions on input size n for which this complexity is as large as $n/\log n$. While exhibiting a function with $\omega(\sqrt{n})$ (standard) OMA complexity is a longstanding open problem, we did not even know of any function with $\omega(\sqrt{n})$ non-trivial-OMA complexity.

Next, we turn to the annotated streaming model, the Prover-Verifier analogue of single-pass data streaming. We reduce from Equals-Index to establish strong lower bounds on the non-trivial complexity (for the analogous notion in this setting) of the fundamental streaming problem of counting distinct items, as well as of graph problems such as k-connectivity (both vertex and edge versions) in a certain edge update model that we call the *support graph turnstile* (SGT) model. To set the benchmark space under which non-trivial annotated streaming schemes should solve these problems, we design classical streaming (sans Prover) algorithms for them under SGT streams by building *strong* ℓ_0-samplers that are robust to such streams and might be of independent interest.

Finally, we exploit graph-theoretic properties to design efficient schemes for k-connectivity in dynamic graph streams. Our schemes for both k-edge and k-vertex connectivity achieve a total cost of $\widetilde{O}(k^2n)$, whereas the optimal streaming complexity for these problems is $\widetilde{\Theta}(kn)$. Additionally, we present a nearly optimal scheme for computing the exact minimum cut with total cost $\widetilde{O}(n)$. This is nearly optimal because the problem has a streaming complexity of $\Omega(n^2)$.

These results together highlight a conceptual separation between classical streaming and annotated streaming: while classical streaming can handle certain SGT streams nearly as efficiently as dynamic streams, annotated streaming cannot.

The content of this chapter is based on a paper co-authored with Prantar Ghosh [GS24], published at ITCS 2024.

Chapter 10: Sublinear-Time Matching Size. We revisit the classical problem of maximum matching. The best-known runtimes for computing an exact maximum matching are $O(m\sqrt{n})$ in general graphs [HK73, MV80], and $m^{1+o(1)}$ in bipartite graphs [CKL+22, VDBCK+23]. When approximation is acceptable, a $(1+\varepsilon)$-approximate matching can be computed in $O(m/\varepsilon)$ time [HK73, DP14]. Yet, as discussed earlier, even linear-time algorithms become inefficient when dealing with massive graphs.

This motivates the study of the maximum matching problem in *sublinear-time*. Unfortunately, strong lower bounds rule out the possibility of sublinear-time algorithms for approximating the maximum matching itself. However, while recovering the matching may be infeasible, estimating its size remains both tractable and valuable. The matching size problem has been extensively studied, with several known upper and lower bounds.

The first wave of results [PR07, NO08, YYI09] focused mainly on bounded-degree graphs and were not sublinear for $\Delta = \Theta(n)$. The second wave of results run in truly sublinear time even on general vertex graph starting with the works [CKK20, KMNFT20]. [Beh22] established a 1/2-approximation in near-linear time, later refined by [BRRS23] to $(1/2+\Omega(1))$-approximation in $O(n^{1+\varepsilon})$ time. A $(2/3-\varepsilon)$-approximation was achieved in $n^{2-\Omega(1)}$ time simultaneously by [BRR23, BKS23b]. This line of work culminated in a $(1, \varepsilon n)$-approximation[4] in $o(n^2)$ time [BKS23a]. These sublinear time upper bounds are complimented by lower bounds which lower bound the query complexity in order to lower bound the running time of any algorithm that approximates the maximum matching size. Recent work by [BRR23] introduced a super-linear lower bound of $\Omega(n^{1.2-o(1)})$ queries for any algorithm that gets a $(2/3+\varepsilon)$-approximation.

[4] An (x, y)-approximation achieves multiplicative error x and additive error y.

However, all known upper and lower bounds are in the adaptive query model, where each query can depend on previous answers. In contrast, non-adaptive query models—where the distribution over all queries must be fixed in advance—are widely studied in property testing, often revealing fundamental gaps between adaptive and non-adaptive complexities. This raises the natural question: is adaptivity also necessary for approximating the maximum matching size in sublinear time? This motivates the goal of achieving a constant or even a polylogarithmic approximation using $\widetilde{O}(n)$ non-adaptive adjacency list queries, similar to what was done by Behnezhad using adaptive queries.

We show that this is not possible by proving that any randomized non-adaptive algorithm achieving an $n^{1/3-\gamma}$-approximation, for any constant $\gamma > 0$, with probability at least $2/3$, must make $\Omega(n^{1+\varepsilon})$ adjacency list queries, for some constant $\varepsilon > 0$ depending on γ. This result highlights the necessity of adaptivity in achieving strong approximations. However, non-trivial upper bounds are still achievable with non-adaptive queries: we present a simple randomized algorithm that achieves an $n^{1/2}$-approximation in $O(n \log n)$ queries.

More generally, we establish a tradeoff between the number of queries and the approximation factor in the lower bound: any n^{δ}-approximation requires $\Omega(n^{1+\varepsilon})$ queries for constants $\varepsilon, \delta > 0$ satisfying $4\varepsilon + 3\delta < 1$. On the upper bound side, we present a generalized algorithm that achieves an n^{δ}-approximation using $O(n^{2-2\delta} \log n)$ non-adaptive queries for any $0 < \delta < 1$.

Together, our results provide the first polynomial upper and lower bounds for estimating maximum matching size using non-adaptive queries, and demonstrate that non-adaptivity imposes strong limitations on the achievable approximation guarantees in this setting.

The content of this chapter is based on a solo-authored manuscript that is currently under submission [Sha25].

1.6 A Reader's Manual

The chapters in this thesis have been designed to be self-contained, with minimal dependencies between them (except for the preliminaries in Chapter 2). As a result, the individual chapters can be read in essentially any order, depending on the reader's interest.

In Chapter 2, we introduce the basic notation and tools that are used throughout this thesis.

The first part of the thesis focuses on optimal graph streaming algorithms. It begins with a chapter on streaming algorithms for the minimum cut problem in insertion-only streams (Chapter 3). The following three chapters address problems in the dynamic

streaming model, covering k-vertex connectivity (Chapter 4), maximum matching and vertex cover (Chapter 5), and finally, the estimation of correlation clustering cost (Chapter 6).

The second part of the thesis explores problems in other modern models of computation. The first chapter studies 4-cycle counting in the dynamic model (Chapter 7). Subsequently, we examine two different oracle-based models. The first is the learning-augmented model, in which we study the maximum independent set problem (Chapter 8). The second is the streaming verification model, where we primarily study k-edge and k-vertex connectivity among other problems (Chapter 9). Finally, the thesis concludes with a chapter on sublinear time algorithms for the matching size problem (Chapter 10).

Chapter 2

Preliminaries

This chapter introduces the notation and models used throughout the thesis, including insertion-only and dynamic graph streaming models. It defines linear sketches and shows their use in streaming algorithms, with tools like $L0$-samplers and sparse recovery. Standard probabilistic tools such as Chernoff and McDiarmid's inequalities, along with limited-independence hash functions, are also presented.

2.1 Notation

For a graph $G = (V, E)$, we use n to represent the number of vertices, m to represent the number of edges, and we write $\mathbf{vec}(E)$ to denote the $\binom{n}{2}$-dimensional vector where $\mathbf{vec}(E)_i$ denotes the multiplicity of the edge e_i in G. We will usually have $V := [n]$, where $[n]$ denotes the set $\{1, 2, \ldots, n\}$. We also use $[a, b]$ for integers $a < b$ to denote the set $\{a, \ldots, b\}$. We use $\deg(v)$ and $N(v)$ for each vertex $v \in V$ to denote the degree and neighborhood of v, respectively. Any vertex v with $\deg(v) = 0$ is called an **isolated** vertex.

For a subset F of edges in E, we use $V(F)$ to denote the vertices incident on F; similarly, for a set U of vertices, $E(U)$ denotes the edges incident on U. We further use $G[U]$ for any set U of vertices to denote the induced subgraph of G on U. We use $|T|$ to denote the number of elements in T when it is a set, and the size of its support when T is a distribution.

Throughout, we use standard asymptotic notation and its "soft" variants, which suppress polylogarithmic factors in the input size n for simplicity of exposition. Specifically, we write $\widetilde{O}(f)$ to denote $O(f \cdot \operatorname{polylog}(n))$, $\widetilde{\Omega}(f)$ to denote $\Omega(f/\operatorname{polylog}(n))$, and $\widetilde{\Theta}(f)$ to

denote functions that are both $\widetilde{O}(f)$ and $\widetilde{\Omega}(f)$. We similarly hide polylogarithmic factors in $\widetilde{o}(f)$ and $\widetilde{\omega}(f)$.

2.2 Streaming and Linear Sketches

We begin with the **insertion-only streaming** model, where the graph evolves through a sequence of edge insertions.

Definition 2.2.1 (Insertion-only graph streams)**.** *An insertion-only stream $\sigma = (\sigma_1, \ldots, \sigma_N)$ defines a multigraph $G = (V, E)$ on n vertices. Each update is a tuple $\sigma_k = (i_k, j_k)$ for $i_k, j_k \in [n]$, representing the insertion of an edge between vertices i_k and j_k.*

The graph is represented by its edge multiplicity vector $\mathbf{vec}(E) \in \mathbb{Z}^{\binom{n}{2}}$, where for each edge (u, v), the corresponding coordinate $\mathbf{vec}(E)_{(u,v)}$ equals the number of times the edge is inserted in the stream. That is,

$$\mathbf{vec}(E)_{(u,v)} = \left|\{k \in [N] : (i_k, j_k) = (u, v)\}\right|.$$

The **dynamic streaming** model generalizes the insertion-only model by allowing both insertions and deletions of edges.

Definition 2.2.2. *A dynamic stream $\sigma = (\sigma_1, \ldots, \sigma_N)$ defines a multi-graph $G = (V, E)$ on n vertices. Each entry of the stream is a tuple $\sigma_k = (i_k, j_k, \Delta_k)$ for $i_k, j_k \in [n]$ and $\Delta_i \in \{-1, +1\}$. The multiplicity of an edge (u, v) is defined as:*

$$\mathbf{vec}(E)_{(u,v)} = \sum_{\sigma_k : i_k = u \,\wedge\, j_k = v} \Delta_k.$$

The multiplicity of every edge is required to be always non-negative.

The goal in this model is to design algorithms that can process a dynamic stream using limited space and at the end of the stream, output a solution to the underlying problem for the (multi-)graph defined by the stream. Throughout the paper, we measure the space of the algorithms in *bits*.

A common objective is to design algorithms that use $\widetilde{O}(n)$ bits of space—known as the **semi-streaming regime**. This memory budget is nearly linear in the number of vertices and can be substantially smaller than the number of edges.

Similar to virtually all other dynamic streaming algorithms, our algorithms will also be based on linear sketches, defined as follows.

Definition 2.2.3 (Linear sketch)**.** *Let* Π *be a problem defined over vectors* $x \in \mathbb{R}^m$ *(e.g., return the* ℓ_2*-norm of* x*). A* linear sketch *for* Π *is an algorithm defined by the following pair:*

- **sketching matrix:** *A matrix* $\Phi \in \text{poly}(m)^{s \times m}$ *that can be chosen randomly and implicitly;*
- **recovery algorithm:** *An algorithm that given the sketching matrix* Φ *and the vector* $\Phi \cdot x$*, returns a solution to* $\Pi(x)$*.*

We refer to the vector $\Phi \cdot x$ *as a* sketch *of* x*, and to the number of bits needed to store* Φ *(implicitly) and* $\Phi \cdot x$ *as the* size *of the linear sketch.*

The linear sketch for an input x *then consists of sampling a sketching matrix* Φ *(independent of* x*), computing the sketch* $\Phi \cdot x$*, and running the recovery algorithm on the sketch to solve the problem.*

For our purpose, we focus on graph problems for the choice of Π in Definition 2.2.3 and then set $x = \mathbf{vec}(E)$ where E is the edge-set of the input graph. The following proposition is well-known.

Proposition 2.2.4. *Let* Π *be a problem defined over vectors* $x \in \mathbb{R}^m$*. Suppose there exists a linear sketch of size* $s(n)$ *for* Π *with probability of success* $p(n)$*. Then, there is also a streaming algorithm for solving* Π *on dynamic streams defining* x *with probability of success* $p(n)$ *using* $O(s(n) + \log m)$ *bits of space.*

Proof. Let Φ be the sketching matrix for the linear sketch. It is enough to show that we can compute $\Phi \cdot x$ for vector x in a dynamic stream as in Definition 2.2.2; the rest then follows by running the recovery algorithm of the sketch at the end on $\Phi \cdot x$. This can be done easily by linearity of the sketch by maintaining $\Phi \cdot x$ and for each update $\sigma_i = (j_i, \Delta_i)$ to x, updating it to

$$\Phi \cdot x + \Phi \cdot \Delta_i \cdot \mathbf{1}_{j_i} = \Phi \cdot (x + \Delta_i \cdot \mathbf{1}_{j_i})$$

($\mathbf{1}_{j_i}$ is the vector that is 1 on j_i-th entry and zero elsewhere)

to get the sketch of the updated vector x. As each update can be computed from the input and (implicit access to) the sketching matrix Φ with additional $O(\log m)$ space for book-keeping, we are done. ∎

Given Proposition 2.2.4 we can simply focus on designing linear sketches for our dynamic streaming problems.

2.3 Standard Tools

We will also use L_0-samplers, a powerful tool used by most dynamic graph streaming algorithms, in our paper. The goal of L_0-samplers is to solve the following basic problem.

Problem 1 (L_0-Sampling)**.** *Given a vector $x \in \mathbb{R}^m$ specified in a dynamic stream, sample x_i uniformly at random from the support of x at the end of the stream.*

We will typically use L_0-samplers by applying them to different pre-specified *subsets* of edges (pairs of vertices) of the underlying graph to sample a uniform edge from those subsets.

Proposition 2.3.1 ([JST11, KNP+17])**.** *There is a linear sketch, called* `L0-Sampler`*, for Problem 1 with size*

$$s_{\texttt{L0}} = s_{\texttt{L0}}(m, \delta_F, \delta_E) = O(\log m \cdot (\log m \cdot \log(1/\delta_F) + \log(1/\delta_E))$$

bits, that outputs FAIL *with probability at most δ_F and outputs a wrong answer with probability at most δ_E.*

We now recall a fundamental problem in communication complexity and state a basic fact about its one-way communication complexity, which we use multiple times in the thesis.

Definition 2.3.2 (Index)**.** *In the IDX_N problem, Alice has a string $x \in \{0.1\}^N$, and Bob has an index $k \in [N]$. The goal is to find $x[k]$, the kth bit of x.*

Proposition 2.3.3 ([Abl96])**.** *The one-way randomized communication complexity $R^{\rightarrow}(\text{IDX}_N) = \Omega(N)$.*

2.4 Probabilistic Tools

Throughout, we will use the term "**with high probability**" (**w.h.p.**) to mean with probability at least $1 - 1/n^c$ for some large constant $c > 0$. The constant c can be made arbitrarily large by only increasing the space of our algorithms by a constant factor thereby remaining within the same asymptotic bounds. Moreover, for our purpose, this probability is large enough that one can always do a union bound over at most $\text{poly}(n)$ different events that we consider; so we do not necessarily mention this each time.

We will also use the term "**with high constant probability**" (**w.h.c.p.**) to mean with probability at least $1 - o(1)$. Additionally, we use "**with constant probability**" (**w.c.p.**) to refer to events that succeed with some fixed probability bounded away from zero (e.g., at least $2/3$ or 0.99), independent of the input size n.

We use the following standard concentration inequalities.

Proposition 2.4.1 (Markov's inequality; c.f. [DP09])**.** *Let X be a non-negative random variable with finite expectation $\mathbb{E}[X]$. Then, for any $a > 0$,*

$$\Pr[X \geqslant a] \leqslant \frac{\mathbb{E}[X]}{a}.$$

Proposition 2.4.2 (Chebyshev's inequality; c.f. [DP09])**.** *Let X be a random variable with finite mean $\mu = \mathbb{E}[X]$ and variance $\sigma^2 = \mathrm{Var}(X)$. Then, for any $t > 0$,*

$$\Pr[|X - \mu| \geqslant t] \leqslant \frac{\sigma^2}{t^2}.$$

We use the following is a standard form of Chernoff bounds:

Proposition 2.4.3 (Chernoff bound; c.f. [DR96, DP09])**.** *Suppose $X_1, \ldots, X_m$ are m independent or negatively associated random variables with range $[0, 1]$ each. Let $X := \sum_{i=1}^{m} X_i$ and $\mu_L \leqslant \mathbb{E}[X] \leqslant \mu_H$. Then, for any $\varepsilon > 0$,*

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

Proposition 2.4.4 (Additive Chernoff bound; c.f. [DP09])**.** *Suppose $X_1, \ldots, X_m$ are m independent random variables with range $[0, 1]$ each. Let $X := \sum_{i=1}^{m} X_i$ and $\mu_L \leqslant \mathbb{E}[X] \leqslant \mu_H$. Then, for any $t > 0$, there is*

$$\Pr(X > \mu_H + t) \leqslant \exp\left(-\frac{2t^2}{m}\right) \quad \text{and} \quad \Pr(X < \mu_L - t) \leqslant \exp\left(-\frac{2t^2}{m}\right).$$

We also need McDiarmid's inequality when there is non-trivial correlation between random variables.

Proposition 2.4.5 (McDiarmid's inequality [M+89])**.** *Let $X_1, \ldots, X_m$ be m independent random variables where each X_i has some range $\mathcal{X}_i$. Let $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \to \mathbb{R}$ be any c-Lipschitz function meaning that for all $i \in [m]$ and all choices of $(x_1, \ldots, x_m), (x'_1, \ldots, x'_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$,*

$$|f(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_m) - f(x_1, \cdots, x_{i-1}, x'_i, x_{i+1}, \cdots, x_m)| \leqslant c.$$

Then, for all $b > 0$,

$$\Pr(|f(X_1, \ldots, X_m) - \mathbb{E}[f(X_1, \ldots, X_m)]| \geqslant b) \leqslant 2 \cdot \exp\left(-\frac{2\,b^2}{m \cdot c^2}\right).$$

Finally, in certain places, we also use limited independence hash functions in our algorithms to reduce their space complexity.

Definition 2.4.6 (Limited-independence hash functions)**.** *For integers $n, m, k \geqslant 1$, a family $\mathcal{H}$ of hash functions from $[n]$ to $[m]$ is called a k-wise independent hash function iff for any two k-subsets $a_1, \ldots, a_k \subseteq [n]$ and $b_1, \ldots, b_k \subseteq [m]$,*

$$\Pr_{h \sim \mathcal{H}}(h(a_1) = b_1 \wedge \cdots \wedge h(a_k) = b_k) = \frac{1}{m^k}.$$

Roughly speaking, a k-wise independent hash function behaves like a totally random function when considering at most k elements. We use the following standard result for k-wise independent hash functions.

Proposition 2.4.7 ([MR95])**.** *For every integers $n, m, k \geqslant 2$, there is a k-wise independent hash function $\mathcal{H} = \{h : [n] \to [m]\}$ so that sampling and storing a function $h \in \mathcal{H}$ takes $O(k \cdot (\log n + \log m))$ bits of space.*

We shall also use the following concentration result on an extension of Chernoff-Hoeffding bounds for limited independence hash function.

Proposition 2.4.8 ([SSS95])**.** *Suppose h is a k-wise independent hash function and $X_1, \ldots, X_m$ are m random variables in $\{0, 1\}$ where $X_i = 1$ iff $h(i) = 1$. Let $X := \sum_{i=1}^{m} X_i$. Then, for any $\varepsilon > 0$,*

$$\Pr(|X - \mathbb{E}[X]| \geqslant \varepsilon \cdot \mathbb{E}[X]) \leqslant \exp\left(-\min\left\{\frac{k}{2}, \frac{\varepsilon^2}{4 + 2\varepsilon} \cdot \mathbb{E}[X]\right\}\right).$$

2.5 Techniques

We now review basic techniques used to establish upper and lower bounds in the streaming model. Notably, the techniques differ significantly between insertion-only and dynamic streams, so we first focus on the insertion-only setting. A common strategy in this setting involves partitioning the stream into chunks, storing a concise summary of each chunk,

and incrementally merging these summaries as the stream progresses. When a new chunk arrives, it is stored in its entirety, processed in an offline fashion, and then merged with the cumulative summary of the previous chunks. The size of each chunk can vary—from a single element to a much larger batch—but it is important to note that the largest chunk size is a lower bound on the amount of space used.

A classic example is the *merge-and-reduce* framework [BS80, MRL99, HPM03, McG14, BDM+20] which is a standard technique for transforming offline algorithms into insertion-only streaming algorithms. In this framework, the stream is divided into chunks of size k, and pairs of consecutive chunks are recursively merged to form summaries of progressively larger parts of the stream, continuing until a single summary of the entire stream is obtained. Since the merging follows a binary tree structure, the recursion has depth only $O(\log n)$, where $\text{poly}(n)$ is the size of the stream, so at most $O(\log n)$ summaries need to be stored at any point. This results in an overall space bound of $\widetilde{O}(k)$. This approach has been successfully applied to construct cut sparsifiers [McG14], quantile summaries [MRL99], and other data structures in streaming settings. We will see the merge and reduce framework in Chapter 3.

Algorithms for dynamic streams differ significantly from those in the insertion-only setting. Since elements can be both inserted and deleted, we cannot simply process the stream by incrementally building summaries. A priori, it is not even clear what kinds of problems we can solve in this setting. Interestingly, despite the challenges of deletions, linear sketches (see Definition 2.2.3) can be implemented in the dynamic streaming model, as demonstrated in Proposition 2.2.4.

Consider a graph $G = (V, E)$, and represent its edge set as a binary vector $\mathbf{vec}(E)$ of dimension $\binom{n}{2} \times 1$. A *sketching matrix* Φ is an $s \times \binom{n}{2}$ matrix (possibly chosen at random), and the product $\Phi\ \mathbf{vec}(E)$ is called the *linear sketch* of the graph. The objective is to design a distribution over such matrices Φ so that the solution to a specific problem on G can be recovered solely from the linear sketch—without any direct access to the original graph.

A trivial solution is to take Φ as the identity matrix, which retains complete information about E. However, the challenge—and the interest—lies in constructing sketches where the number of rows $s \ll m$. Note that $\mathbf{vec}(E)$ is a binary vector, so in theory, one could reduce the sketching matrix Φ to one row ($s = 1$) by assigning weights 2^i to the i^{th} column of Φ, effectively encoding the entire vector $\mathbf{vec}(E)$ in the sketch as an integer. However, this approach requires entries in Φ to be exponentially large, which is impractical. Hence, we restrict the entries of Φ to be bounded by $\text{poly}(n)$.

In fact, for sufficiently long dynamic streams, any streaming algorithm can be refor-

mulated as a linear sketch [LNW14]. As a result, most dynamic streaming algorithms in the literature are based on linear sketches, with many relying on `L0-Sampler` as a key primitive. An `L0-Sampler` is a linear sketch that returns a uniformly random element with non-zero frequency in the stream (Problem 1). Most dynamic streaming algorithms—like those for spanning forest, minimum cut, cut sparsifiers, vertex connectivity, maximum matching, and minimum vertex cover—are built upon this foundational tool. We will see examples of this in Chapters 4 to 6.

On the lower-bound side, we rely on the well-established framework of communication complexity. In this model, two computationally unbounded players—Alice with input x and Bob with input y—collaborate to compute a function $f(x, y)$. The cost is measured solely by the number of bits exchanged between them, not by their internal computation. For randomized protocols, they are also given access to a shared source of randomness.

To prove lower bounds in single-pass streams, we consider the more restrictive setting of *one-way* communication, where Alice sends Bob a message about her input. Bob then uses the message along with his input to compute the output. Lower bounds in the communication model translate directly into streaming lower bounds, as streaming algorithms can be viewed as one-way communication protocols. Specifically, a streaming algorithm A using s bits of space can be converted into a communication protocol P with communication cost s: Alice runs algorithm A on her part of the input, treating it as a stream, and sends the memory state (a message of size s) to Bob, who continues running A on his part of the input and produces the final output.

The most common communication problem used for proving lower bounds is the *Index* problem, where Alice has a binary string of length N, and Bob has an index $i \in [N]$, and their goal is to compute $x[i]$. This problem has been used to establish lower bounds for a wide range of graph problems, including connectivity, eulerian testing, bipartiteness, cycle-freeness, minimum spanning tree, k-edge connectivity, k-vertex connectivity, and constant-factor approximate minimum cut [SW15]. We will use the Index problem to prove lower bounds in Chapters 3, 6 and 9.

Part I

Part I: Optimal Graph Streaming Algorithms

Chapter 3

Streaming Minimum Cut

3.1 Introduction

The *minimum cut* of an undirected graph $G = (V, E)$, with n vertices and m edges, is a subset of vertices $S \subset V$ such that the number (or weight in a weighted graph) of edges with one endpoint in S and the other in $V \setminus S$ is minimized. The minimum cut problem has been studied extensively on its own both as a fundamental graph problem and in relation to other important problems such as edge-connectivity and network flows. Besides being theoretically interesting, finding minimum cuts is also a problem of practical interest. For example, they allow for the calculation of social network metrics such as influence [WDL+20] and are used to quantify the robustness of power networks [HJJ+14] and road networks [SK23] to failures.

Karger [Kar00] gave the first near-linear time randomized algorithm for computing the minimum cut of a graph. This naturally raised the question of whether a deterministic algorithm with similar guarantees could be achieved. [LP20] showed that the minimum cut problem can be deterministically reduced to computing polylogarithmic many maximum flows. The recent breakthrough in maximum flow solved the problem in $m^{1+o(1)}$ time deterministically [VDBCK+23, CKL+22], implying that the minimum cut can also be computed deterministically in $m^{1+o(1)}$ time. Finally, a line of work [KT18, HRW20, Li21, HLRW24] derandomized Karger's approach, ultimately resulting in a deterministic $\widetilde{O}(m)$-time algorithm for finding the minimum cut.

We study the minimum cut problem in the insertion-only streaming model. It is known that computing the exact minimum cut in this model requires $\Omega(n^2)$ space [Zel11]. This motivates the question of whether good approximations are possible within semi-streaming

space. The best known upper bound in insertion-only streams for $(1+\varepsilon)$-approximate minimum cut was given by Ahn and Guha [AG09], using $\widetilde{O}(n/\varepsilon^2)$ space. This gives a semi-streaming algorithm when $\varepsilon = \Theta(1)$ (or even when $\varepsilon = 1/\text{polylog}\, n$). Sun and Woodruff [SW15] showed that this result is essentially tight by proving a lower bound of $\Omega(n \log n)$ bits for any constant-factor approximation.

The key method so far for solving minimum cut with low-memory is the usage of *cut sparsifiers*. This approach was also used by [AG09] in their streaming upper bound. The notion of cut sparsifiers was introduced by Benczur and Karger [BK96] and has been extremely influential. Given a weighted graph $G = (V, E, w)$ with polynomially bounded edge weights $w : E \to \mathbb{R}_+$, and an accuracy parameter $\varepsilon > 0$, a cut sparsifier of G is a weighted graph H on the same vertex set V, such that the weight of every cut in G is $(1+\varepsilon)$-approximated by the weight of the corresponding cut in H.

Specifically, [BK96] shows that a cut sparsifier always exists with $O(n \log n/\varepsilon^2)$ edges. This bound was improved to $O(n/\varepsilon^2)$ edges by [BSS12]. These construction has had a tremendous impact on cut problems in graphs, see, e.g., [BK96, BK15, KL02, She09, Mad10]. [Alo97, ACK+16, CKST19] proved a lower bound of $\Omega(n(\log n)/\varepsilon^2)$ bits (even for data structures that preserve approximate values of all cuts). This also settled the space complexity of constructing cut sparsifiers in the streaming model up to polylogarithmic factors, yielding a nearly tight bound of $\widetilde{\Theta}(n/\varepsilon^2)$. However, this lower bound does not apply to minimum cut. [AG09] also proved a lower bound of $\Omega(n \log \frac{1}{\varepsilon})$ bits for a $(1+\varepsilon)$ approximation to the minimum cut problem, although they conjectured that their upper bound of $\widetilde{O}(n/\varepsilon^2)$ bits was optimal for approximate minimum cut and other cut related problems. We refute this conjecture by presenting a more space-efficient algorithm:

Theorem 1. *There is a one-pass insertion-only streaming algorithm that, with high probability, computes an $(1+\varepsilon)$-approximation of the minimum cut on weighted graphs using $\widetilde{O}(n/\varepsilon)$ bits of space.*

This result reveals, somewhat surprisingly, that approximating the minimum cut is strictly easier than constructing a for-all cut sparsifier in the streaming model. Our approach leverages a for-each cut sparsifier as the key technical tool.

To overcome the limitations of for-all cut sparsifiers—particularly the prohibitive $1/\varepsilon^2$ dependence for very small ε—Andoni et al. [ACK+16] introduced a relaxed formulation known as *for-each* cut sparsifiers. Instead of approximating all cuts simultaneously, the goal is to construct a data structure sk_G such that, for every *fixed* cut $S \subseteq V$, the estimate $\text{sk}_G(S)$ approximates the true cut value within a $(1+\varepsilon)$ factor, with probability at least $2/3$[1].

[1]This probability can be amplified to high probability via $O(\log n)$ independent repetitions and taking

Note that the value of every cut has to be approximately preserved with high probability, but this guarantee does not have to hold simultaneously for all cuts.

Surprisingly, [ACK+16] showed that such a data structure exists in $\widetilde{O}(n/\varepsilon)$ space for cut sparsification with poly(n)-bounded integer edge weights. They also showed that this is optimal up to polylogarithmic factors. In a follow-up work, [CGP+18] showed that such a sketch can be chosen as a reweighted subgraph of G. However, it was not known how to construct a for-each cut sparsifier in a stream in better than $\widetilde{O}(n/\varepsilon^2)$ space, which follows from computing a for-all cut sparsifier. A natural question is whether we can implement a streaming algorithm that computes a for-each sparsifier within using similar space. We answer this question in the affirmative by proving the following:

Theorem 2. *There is a one-pass insertion-only streaming algorithm that, with high probability, constructs a $(1+\varepsilon)$ for-each cut sparsifier of weighted graphs using $\widetilde{O}(n/\varepsilon)$ bits of space.*

We know this is nearly optimal because [ACK+16] proved an offline lower bound of $\Omega(n/\varepsilon)$ bits for such sparsifiers. However, no analogous lower bound was known for the minimum cut problem, leaving open the possibility that the existing upper bound could be further improved or that the known $\Omega(n \log \frac{1}{\varepsilon})$ lower bound could be strengthened. We show the latter by proving a nearly tight lower bound:

Theorem 3. *Fix $\varepsilon > 1/n$. Any randomized algorithm that outputs a $(1+\varepsilon)$-approximation to the minimum cut of a simple, undirected graph in a single pass over a stream with probability at least $2/3$ requires $\Omega(n/\varepsilon)$ bits of space. If the algorithm is deterministic and $\varepsilon \geqslant 1/n^{1/4}$, then the algorithm requires $\Omega(n/\varepsilon^2)$ bits of space.*[2]

This result settles the space complexity of approximating the minimum cut in insertion-only streams up to polylogarithmic factors. It also establishes a separation between randomized and deterministic algorithms, as the deterministic complexity remains at $\widetilde{\Theta}(n/\varepsilon^2)$ bits[3]. In summary, our work resolves the complexity of both exact and approximate minimum cut in the insertion-only model for both randomized and deterministic settings.

But what if we want to compute the minimum cut exactly using $o(n^2)$ space? Can we somehow relax the model to achieve this? If the algorithm is allowed two passes over the stream, the exact minimum cut can be found using only $\widetilde{O}(n)$ bits of space for unweighted

the median estimate. This also lets us estimate the cut value for poly(n) cuts simultaneously.

[2]This latter assumption on ε is the same as used in [CKST19].

[3]The algorithm of [AG09] can be made deterministic.

graphs [RSW18, AD21]. For weighted graphs, [MN20] solves the problem in $\log n$ passes. However, many streaming applications require solving the problem in exactly one pass because the stream cannot be stored. To overcome the $\Omega(n^2)$ space barrier of adversarial-order streams in a single pass, we turn to a natural relaxation known as the *random-order* model. In this model, the graph can still be chosen adversarially, but its edges arrive in the stream in a random order.

The motivation for studying this model stems from the fact that real-world data is typically not adversarial. Adversarial streams are adversarial in both their input and arrival order, while random-order streams relax the arrival order, making them more representative of real-world data. Additionally, the random-order streaming model may help explain the empirical performance of certain heuristics.

The random-order model is known to be provably easier than the adversarial-order model. For instance, there are established separations for problems like maximum matching [Kap21, Ber24]. But does such a separation exist for minimum cut as well? We answer this question in the affirmative by proving a strong separation between adversarial and random-order streams for unweighted graphs:

Theorem 4. *There exists a one-pass insertion-only streaming algorithm in the random-order model that with high probability outputs all minimum cuts $(S, V \setminus S)$, for a simple, unweighted graph using $\widetilde{O}(n)$ bits of space.*

We note that Theorem 4 computes the exact minimum cut using only $\widetilde{O}(n)$ space in a single pass over a random-order stream—an outcome that we find surprisingly achievable. This adds to a growing list of unexpected results for minimum cut in the streaming model. Previously, a two-pass exact algorithm was known in adversarial-order streams [AD21], whereas any single-pass algorithm in this setting requires $\Omega(n^2)$ memory [Zel11]. Moreover, since even determining whether a graph is connected in a random-order stream requires $\Omega(n)$ space [CCM16], our $\widetilde{O}(n)$-space algorithm for computing the exact minimum cut is optimal up to polylogarithmic factors—even if the goal is only to output the value. Notably, our algorithm can also recover all edges crossing the cut, provided the cut is non-singleton (i.e., both sides contain at least two vertices).

We list the current state-of-the-art results for single-pass minimum cut streaming algorithms in Table 3.1. Our main result is an $\widetilde{O}(n/\varepsilon)$ space streaming algorithm that finds a $(1+\varepsilon)$-approximate minimum cut in a single-pass stream.

Related Work. The random-order model was first studied by [MP80] for sorting in limited space. Many subsequent works have studied problems in random-order streams

Stream Type	Exact/Approx $(1+\varepsilon)$	Upper Bound	Lower Bound
Adversarial	Exact	$O(n^2)$ (full graph)	$\Omega(n^2)$ [Zel11]
Adversarial	Approx, Deterministic	$\widetilde{O}(n/\varepsilon^2)$ [BSS12]	$\Omega(n/\varepsilon^2)^*$ (Theorem 3)
Adversarial	Approx, Randomized	$\widetilde{O}(n/\varepsilon)$ (Theorem 1)	$\Omega(n/\varepsilon)$ (Theorem 3)
Random-Order	Exact	$\widetilde{O}(n)$ (Theorem 4)	$\Omega(n)$ [CCM16]

Table 3.1: Minimum Cut Space Complexity in the Single-Pass Insertion-Only Streaming Setting

[GM09, KKS14a, CCM16, MMPS17, PS18, CFPS19, CGMV20], some of which show a clear separation between the adversarial and random-order models.

One notable line of work focuses on maximum matching within this model [KMM12, GKMS19, Kon18a, ABB+19, FHM+20, Ber24, AB21, HWK24]. The best-known upper bound in the random order model [AB21] gets a slightly better than 2/3-approximation in semi-streaming space, which is better than the best possible $\frac{1}{1+\ln 2} \approx 0.59$-approximation in the adversarial model due to the lower bound of [Kap21], showing another separation between adversarial and random-order streams.

3.1.1 Our Techniques

Approximate Minimum Cut. Suppose that H is a $(1+\varepsilon)$-for-all sparsifier of G. Then the minimum cut of H is a $(1+\varepsilon)$-approximation to that of G. However, such a conclusion will not hold if H is instead a $(1+\varepsilon)$ for-each sparsifier, as the total number of cuts is exponential. Thus, storing just a for-each cut sparsifier in the stream is not enough to find the minimum cut. Fortunately the number of 1.5-approximate minimum cuts is $O(n^3)$ ([Kar00]). Hence, if we run a for-all sparsifier algorithm in parallel with accuracy $\varepsilon' = 0.5$, we can obtain a list of $O(n^3)$ candidate minimum cuts in G. We can then use the for-each sparsifier to check the values of these cuts and find the minimum one. The for each sparsifier preserves the values of all these cuts up to a factor of $(1+\varepsilon)$ by a union bound over all $O(n^3)$ candidate cuts. Thus, the minimum one will be a $(1+\varepsilon)$-approximate minimum cut.

For the lower bound, we consider the k-edge-connectivity problem. This problem asks whether the minimum cut of a graph is $< k$ or $\geqslant k$. Suppose that there is an algorithm that solves the $(1+\varepsilon)$-approximate minimum cut value problem. Then, we can use it to

solve the k-edge-connectivity problem with $k = O(1/\varepsilon)$. Combining with the $\Omega(kn)$ bit space lower bound for k-edge connectivity [SW15], it follows that there is an $\Omega(n/\varepsilon)$ bit space lower bound for approximate minimum cut.

For-Each Cut Sparsifier. The main difficulty in implementing the algorithms of [ACK+16] and [JS18] in the streaming setting is that both algorithms require careful graph decomposition. Take the algorithm in [ACK+16] as an example: it partitions the graph into several components, where each component is well-connected and does not have a sparse cut smaller than $1/\varepsilon$. In the worst case, such a procedure may take $\log n$ levels recursively, which seems unachievable in a single-pass stream. Besides, since the for-each sketch of both algorithms is not a graph, it is unclear how to merge two sketches directly. Indeed, natural ways of merging such sketches may destroy the decomposition into sparse cuts. To address these issues, we instead consider the work of [CGP+18] in which the authors show how to construct a for-each sparsifier which is, in fact, a re-weighted subgraph of the input graph, with $\widetilde{O}(n/\varepsilon)$ edges. Combining this and the merge-and-reduce framework gives a for-each cut sparsifier with $\widetilde{O}(n/\varepsilon)$ edges in a single-pass stream.

Exact Minimum Cut in Random-order Streams. Recall that we assume that the graph is simple and unweighted, and each edge arrives in a random order stream. We also assume the minimum cut size is $\Omega(\log n)$. If the size is smaller, a for-all cut sparsifier with accuracy parameter $\varepsilon' = 1/\log^2 n$ preserves the exact value of the minimum cut. We also assume that the minimum cut is non-singleton because storing the minimum degree would give the singleton minimum cut. The main idea here is that by looking at the prefix of edges in a stream, which we will call the graph H, we roughly learn the sizes of all the cuts up to a small constant factor. Using this information, we learn all 1.1-approximate minimum cuts in G. The next question is how to get the exact value of all these cuts (so we can find the minimum one). For the cut edges in H, note that at this time, the cut values of all these cuts in H is $\Theta(\log n)$, and hence a for-all sparsifier of H with accuracy parameter $\varepsilon' = 1/\log^2 n$ preserves the exact value of these cuts in H. We note that the total number of edges that participate in at least one of the 1.1-approximate non-singleton cuts is $O(n)$ [RSW18], which is interesting because there could be as many as $O(n^2)$ such cuts. Hence, we can store all these $O(n)$ edges to get the exact cut values of the approximate minimum cuts in $G \setminus H$. Putting the two things together, we obtain the exact value of all 1.1-approximate minimum cuts and thus obtain the exact minimum cut value in G (along with all the minimum cuts).

3.2 Preliminaries

Notation. $S \subset V$ denotes a cut between non-empty sets S and $V \setminus S$. We use $\lambda_G(S)$ to denote the weight of cut S, which is defined as the sum of the weights of all edges that cross the cut i.e. edges that have one endpoint in S and the other in $V \setminus S$. We denote by λ_G, the weight of the minimum cut of G. We will drop the subscript when the graph is clear from context. Consider the following definition of for-all cut sparsifiers:

Definition 3.2.1 (For-all Sparsifier)**.** *We say that a $(1+\varepsilon)$ for-all cut-sparsifier is a weighted subgraph H of G such that*

$$(1-\varepsilon)\lambda_G(S) \leqslant \lambda_H(S) \leqslant (1+\varepsilon)\lambda_G(S)$$

holds with high probability for all non-empty sets $S \subset V$.

If we mention a cut sparsifier without mentioning the kind then it is a for-all cut sparsifier by default. We next define for-each cut sparsifiers:

Definition 3.2.2 (For-each Sparsifier)**.** *We say that a $(1+\varepsilon)$ for-each cut-sparsifier is a weighted subgraph H of G such that*

$$(1-\varepsilon)\lambda_G(S) \leqslant \lambda_H(S) \leqslant (1+\varepsilon)\lambda_G(S)$$

holds for each $S \subset V$ with high probability.

A for-all cut sparsifier has to, with high probability, approximately preserve the value of all cuts. On the other hand, a for-each cut sparsifier has to, with high probability, approximately preserve the value of just a single cut. As a consequence of union bound, we can guarantee that a polynomial number of cuts will have their values approximately preserved by a for-each cut-sparsifier.

For the sake of convenience and clarity in our analysis, we adopt the use of a $(1+\varepsilon)^k$ sparsifier, denoting that the cut values are preserved with an overestimate of at most a factor of $(1+\varepsilon)^k$ and an underestimate of at most a factor of $(1-\varepsilon)^k$.

We now state some useful results. The first is an insertion-only streaming algorithm for a for-all cut-sparsifier.

Proposition 3.2.3 ([AGM12b])**.** *There exists an insertion-only streaming algorithm that returns a $(1+\varepsilon)$ for-all cut-sparsifier of a graph with $\mathrm{poly}(n)$ edge weights using $\widetilde{O}(\frac{n}{\varepsilon^2})$ space.*

The following gives a bound on the number of approximate minimum cuts.

Definition 3.2.4. *An α-minimum cut is a cut of value at most α times the value of the minimum cut.*

Proposition 3.2.5 ([KS96])**.** *In any weighted graph, the number of α-minimum cuts is $\leqslant (2n)^{2\alpha}$.*

The following result shows the existence of small space for-each cut-sparsifiers.

Proposition 3.2.6 ([CGP+18])**.** *There exists an algorithm $\mathcal{K}$ that when given a weighted graph G and an approximation factor ε returns a $(1+\varepsilon)$ for-each cut-sparsifier of G with $\widetilde{O}(n/\varepsilon)$ edges.*

We now cite an important lower bound on k-edge-connectivity.

Proposition 3.2.7 ([SW15])**.** *Any randomized algorithm solving the k-edge-connectivity problem in an insertion-only stream needs at least $\Omega(kn)$ bits.*

3.3 Streaming For-each Sparsifier

In this section, we give a streaming algorithm for a for-each cut-sparsifier of weighted graphs using $\widetilde{O}(n/\varepsilon)$ space, thus proving Theorem 2. We start by showing two important properties of the for-each cut-sparsifier. First, the union of for-each sparsifiers of two disjoint graphs gives a for-each sparsifier of their union.

Lemma 3.3.1. *(Mergeability) Let H_1 and H_2 be $(1+\varepsilon)$ for-each sparsifiers of edge-disjoint graphs G_1 and G_2 on the same set of vertices. Then, $H_1 \cup H_2$ is a $(1+\varepsilon)$ for-each sparsifier of graph $G_1 \cup G_2$.*

Proof. Consider an arbitrary cut S in G. Since H_1 is a $(1+\varepsilon)$ for-each sparsifier of G_1, we have:

$$(1-\varepsilon)\lambda_{G_1}(S) \leqslant \lambda_{H_1}(S) \leqslant (1+\varepsilon)\lambda_{G_1}(S).$$

Since H_2 is a $(1+\varepsilon)$ for-each sparsifier of G_2, we have:

$$(1-\varepsilon)\lambda_{G_2}(S) \leqslant \lambda_{H_2}(S) \leqslant (1+\varepsilon)\lambda_{G_2}(S).$$

Adding the above two equations gives us:

$$(1-\varepsilon)(\lambda_{G_1}(S)+\lambda_{G_2}(S)) \leqslant \lambda_{H_1}(S)+\lambda_{H_2}(S) \leqslant (1+\varepsilon)(\lambda_{G_1}(S)+\lambda_{G_2}(S).$$

Using the fact that the graphs are edge-disjoint implies $\lambda_{G_1 \cup G_2}(S) = \lambda_{G_1}(S) + \lambda_{G_2}(S)$:

$$(1-\varepsilon)\lambda_{G_1 \cup G_2}(S) \leqslant \lambda_{H_1 \cup H_2}(S) \leqslant (1+\varepsilon)\lambda_{G_1 \cup G_2}(S).$$

Thus, $H_1 \cup H_2$ is a $(1+\varepsilon)$ for-each sparsifier of $G_1 \cup G_2$. Note that all these equations hold with high probability. ∎

We now show the second important property which says that a sparsifier of a sparsifier is also a sparsifier.

Lemma 3.3.2. *(Composability) Let H_1 be a $(1+\varepsilon)$ for-each sparsifier of G and H_2 be a $(1+\varepsilon)$ for-each sparsifier of H_1. Then, H_2 is a $(1+\varepsilon)^2$ for-each sparsifier of G.*

Proof. Consider an arbitrary cut S in G. Since H_1 is a $(1+\varepsilon)$ for-each sparsifier of G, the following holds with high probability:

$$(1-\varepsilon)\lambda_G(S) \leqslant \lambda_{H_1}(S) \leqslant (1+\varepsilon)\lambda_G(S)$$

Since H_2 is a $(1+\varepsilon)$ for-each sparsifier of H_1, we have the following:

$$(1-\varepsilon)\lambda_{H_1}(S) \leqslant \lambda_{H_2}(S) \leqslant (1+\varepsilon)\lambda_{H_1}(S)$$

By composition, we have:

$$(1-\varepsilon)^2\lambda_G(S) \leqslant (1-\varepsilon)\lambda_{H_1}(S) \leqslant \lambda_{H_2}(S) \leqslant (1+\varepsilon)\lambda_{H_1}(S) \leqslant (1+\varepsilon)^2\lambda_G(S)$$

We conclude that H_2 is a $(1+\varepsilon)^2$ for-each sparsifier of G. ∎

Now, with these tools, we will prove Theorem 2. We divide the incoming stream into t chunks of n/ε edges. We denote the chunks as G_i^0 for $i \in [t]$. We also define $\log t$ levels and recursively define a graph on level j using two graphs from level $j-1$: $G_i^j := G_{2i-1}^{j-1} \cup G_{2i}^{j-1}$ for $j \in [\log t]$. We also define a series of sparsified graphs corresponding to each of the G_i^js as follows:

$$H_i^0 := G_i^0, \quad H_i^j := \mathcal{K}(H_{2i-1}^{j-1} \cup H_{2i}^{j-1})$$

where $\mathcal{K}$ is the algorithm from Proposition 3.2.6 and it is run with parameter $\varepsilon' = \varepsilon/2\log t$. Consider the full description of the algorithm:

Algorithm 1. For Each Sparsifier

Input: A weighted graph G in an insertion-only stream and an approximation parameter ε.

Output: A graph H which is a $(1+\varepsilon)$ for-each cut-sparsifier of G with $\widetilde{O}(n/\varepsilon)$ edges.

1. Divide stream into $t := m\varepsilon/n$ chunks: $G_0^0, \ldots, G_{t-1}^0$, each with n/ε edges.
2. As stream chunk G_i^0 comes in do the following:
   - Store $H_i^0 := G_i^0$.
   - Check if any level $j \in [\log t]$ has at least 2 sparsifiers $H_{2i'}^j$ and $H_{2i'+1}^j$.
   - If so store graph $H_{i'}^{j+1} = \mathcal{K}(H_{2i'}^j \cup H_{2i'+1}^j)$ and then delete $H_{2i'}^j$ and $H_{2i'+1}^j$.
3. Return $H = H_0^{\log t}$

We first show that H_i^j is a $(1+\varepsilon')^j$ for-each sparsifier of G_i^j.

Claim 3.3.3. *For every i, j H_i^j is a $(1+\varepsilon')^j$ for-each sparsifier of corresponding graph G_i^j.*

Proof. We prove this by induction on j.

Base Case: Let $j = 0$. The claim asserts that H_i^0 is a $(1+\varepsilon')^0 = 1$ for-each sparsifier of the subtree below it. The corresponding graph of H_i^0 is just G_i^0. By definition, $H_i^0 = G_i^0$ so H_i^0 gives a 1-approximation.

Inductive Step: Suppose the claim holds for $j = j'$. In other words, for all i, $H_i^{j'}$ is a $(1+\varepsilon')^{j'}$ sparsifier of its corresponding graph $G_i^{j'}$. Now, consider $H_{i'}^{j'+1}$. By definition, $H_{i'}^{j'+1}$ is a sparsifier of the union of $H_{2i'}^{j'}$ and $H_{2i'+1}^{j'}$, each of which are $(1+\varepsilon')^{j'}$ sparsifiers for their corresponding graphs. By Lemmas 3.3.1 and 3.3.2, $H_{2i'}^{j'} \cup H_{2i'+1}^{j'}$ is a $(1+\varepsilon')^{j'}$ sparsifier for $G_{2i'}^{j'} \cup G_{2i'+1}^{j'} = G_{i'}^{j'+1}$. So, $H_{i'}^{j'+1} = \mathcal{K}(H_{2i'}^{j'} \cup H_{2i'+1}^{j'})$ is a $(1+\varepsilon')^{j'+1}$ of $G_{i'}^{j'+1}$ sparsifier.

Thus, for every i, j, H_i^j is a $(1+\varepsilon')^j$ for-each sparsifier of G_i^j. ∎

We now prove the correctness guarantee of the algorithm.

Claim 3.3.4. *The output H of Algorithm 1 is a $(1+\varepsilon)$ for-each cut-sparsifier of G.*

Proof. Observe that $G_0^{\log t} = G$ and $H_0^{\log t}$ is a $(1+\varepsilon')^{\log t}$ for-each cut-sparsifier of G (Claim 3.3.3). To prove that $H_0^{\log t}$ is a $(1+\varepsilon)$ for-each sparsifier, we need to show that $(1+\varepsilon')^{\log t} = (1+\frac{\varepsilon}{2\log t})^{\log t} \leqslant (1+\varepsilon)$.

$$\begin{aligned}\left(1+\frac{\varepsilon}{2\log t}\right)^{\log t} &\leqslant \exp\left(\frac{\varepsilon}{2\log t}\cdot\log t\right) && (1+x\leqslant e^x)\\ &= \exp(\varepsilon/2)\\ &\leqslant (1+\varepsilon) && (x\in[0,1], e^x\leqslant 1+2x)\end{aligned}$$

Thus, H is a $(1+\varepsilon)$ for-each cut-sparsifier of G. ∎

We finally bound the space taken by the algorithm.

Claim 3.3.5. *Algorithm 1 uses $\widetilde{O}(n/\varepsilon)$ space.*

Proof. Observe that after we use two sparsifiers, $H_{2i'}^{j'}$ and $H_{2i'+1}^{j'}$, to construct $H_{i'}^{j'+1}$, we can delete them. In general, we will have to store at most two sparsifiers at each level at any given time implying that the space of a level is $\widetilde{O}(n/\varepsilon)$. Thus, the overall space usage is $\widetilde{O}(n/\varepsilon)\cdot\log t = \widetilde{O}(n/\varepsilon)$ (since $\log t \leqslant \log n$). ∎

Claims 3.3.4 and 3.3.5 together prove Theorem 2.

We note that this construction also gives a spectral sketch in $\widetilde{O}(n/\varepsilon)$ space because the algorithm of [CGP+18] in Proposition 3.2.6 gives a for-each spectral sparsifier (not just a for-each cut-sparsifier). [CGP+18] also gives a fast algorithm to compute this spectral sketch ($m^{1+o(1)}$ time) with a small overhead in space ($n^{o(1)}$ multiplicative factor). This implies that our streaming algorithm can be made to run in $m^{1+o(1)}$ time if the space used is $n^{1+o(1)}/\varepsilon$. We do not discuss these ideas in detail because our main focus is to find the approximate minimum cut which we show in the next section.

3.4 Streaming Algorithm for Approximate Minimum Cut

We present our single-pass streaming algorithm for approximate minimum cut in this section. Consider the following theorem:

Theorem 5. *There is an algorithm that given a weighted graph G in an insertion-only stream outputs a $(1+\varepsilon)$-approximate minimum cut of G with high probability using $\widetilde{O}(n/\varepsilon)$ space.*

The idea is to run a for-each sparsifier with parameter ε and a for-all sparsifier with a small constant parameter in parallel. After the stream, we use the for-all sparsifier to find all the small cuts and then estimate the value of those cuts more precisely using the for-each sparsifier and thus finding an approximate minimum cut. The following algorithm formalizes this idea:

Algorithm 2. Approximate Minimum Cut

Input: A graph G in an insertion-only stream, and an approximation parameter ε

Output: A $(1+\varepsilon)$-approximation to the minimum cut value in G.

1. Construct a 1.1 for-all cut-sparsifier of G using Proposition 3.2.3.
2. Run Algorithm 1 to obtain H, a $(1+\varepsilon)$ for-each cut-sparsifier of G.
3. Use the 1.1 cut-sparsifier to find all 1.1-approximate min-cuts.
4. Query sparsifier H on all the 1.1-approximate min-cuts and return smallest cut.

We prove this theorem via two lemmas. The first shows that the algorithm is correct.

Lemma 3.4.1. *Algorithm 2 returns a $(1+\varepsilon)$-approximate minimum cut with high probability.*

Proof. By Proposition 3.2.5, we know that there are at most $O(n^{2\alpha}) = O(n^{2.2})$ 1.1-approximate min-cuts. The minimum cut of G will be included in this set of approximate cuts. With high probability, H gives a $(1+\varepsilon)$-approximation to all these cuts by a union bound over the failure probabilities. Thus, we find a $(1+\varepsilon)$-approximate min-cut. ∎

The second lemma bounds the space of this algorithm.

Lemma 3.4.2. *Algorithm 2 uses $\widetilde{O}(n/\varepsilon)$ space.*

Proof. By Proposition 3.2.3 constructing the 1.1 cut-sparsifier takes $O(n)$ space. By Theorem 2, we use $\widetilde{O}(n/\varepsilon)$ space for the for-each sparsifier. Thus, overall we use $\widetilde{O}(n/\varepsilon)$ space. ∎

As a consequence of Lemmas 3.4.1 and 3.4.2, we obtain Theorem 5.

3.5 Approximate Minimum Cut Lower Bounds

To prove this lower bound, we consider the k-edge-connectivity problem as follows. The *edge-connectivity* of a graph is the minimum number of edges that need to be deleted to disconnect the graph. The k-edge-connectivity problem asks whether the edge connectivity of a graph is $< k$ or $\geqslant k$. The work of [SW15] shows that any deterministic algorithm that solves the k-edge-connectivity problem in insertion-only streams needs $\Omega(kn \log n)$ bits of space. We note that the proof in [SW15] also works for randomized algorithms with success probability at least $1 - 1/n$, which implies a lower bound of $\Omega(kn)$ bits of space for any randomized algorithm with constant probability of success. We now prove the randomized and deterministic lower bounds of Theorem 3. We start with the randomized lower bound:

Lemma 3.5.1. *Fix $\varepsilon > 1/n$. Any randomized algorithm that outputs a $(1+\varepsilon)$-approximation to the minimum cut of a simple, undirected graph in a single pass over a stream with probability at least $2/3$ requires $\Omega(n/\varepsilon)$ bits of space.*

Proof. Consider the lower bound for k-edge-connectivity in [SW15] with $k = \frac{1}{10\varepsilon}$ (Proposition 3.2.7). Distinguishing between whether the graph is $< k$-connected or $\geqslant k$-connected needs $\Omega(kn) = \Omega(n/\varepsilon)$ space. This implies that deciding whether the min-cut has value $< k$ or $\geqslant k$ also requires $\Omega(kn) = \Omega(n/\varepsilon)$ space. Suppose that there is a $(1+\varepsilon)$-approximation algorithm for the minimum cut value, in the case where the edge connectivity $\lambda < k$, the approximate value is at most $(1+\varepsilon)\lambda \leqslant (1+1/10k)\cdot(k-1) < k - 9/10$. In the case where the connectivity $\lambda \geqslant k$, the approximate value is at least $(1-\varepsilon)\lambda \geqslant (1-1/10k)\cdot k > k - 1/10$. This implies that an algorithm that gives a $(1+\varepsilon)$-approximation to minimum cut value, solves k-edge-connectivity for $k = \frac{1}{10\varepsilon}$. Thus, we obtain a lower bound of $\Omega(n/\varepsilon)$ bits of space for $(1+\varepsilon)$-approximating the value of the minimum cut. ∎

In addition to the proof above that considers the k-edge-connectivity problem, we also provide an alternative self-contained construction for the randomized result using the Index communication problem in Section 3.6.

Deterministic Lower Bound. If the minimum cut value algorithm is deterministic, then we deterministically solve k-edge-connectivity for $k = \frac{1}{10\varepsilon}$. This gives us a lower bound of $\Omega(kn \log n) = \Omega(n \log n/\varepsilon)$ bits of space for deterministic algorithms. We show how to improve this lower bound to $\Omega(n/\varepsilon^2)$ bits for $\varepsilon \geqslant n^{-1/4}$. Note that this assumption on ε is consistent with the previous work of [CKST19]. They also use this assumption on ε and prove a lower bound of $\Omega(n \log n/\varepsilon^2)$ bits for cut sparsifiers. We need this assumption

because we are trying to simulate a for-each sparsifier using a deterministic algorithm for minimum cut. This simulation only works when the value of the queried cut is at most n. The hard instance of [ACK+16] has queried cut values of at most $1/\varepsilon^4$ implying that $\varepsilon \geqslant n^{-1/4}$. Also, note that $\varepsilon \geqslant n^{-1/2}$ because this already gives a lower bound of $\Omega(n^2)$. One way to bridge this gap would be to come up with a better hard instance with queried cut values at most $1/\varepsilon^2$.

Lemma 3.5.2. *Fix $\varepsilon > 1/n^{1/4}$. Any deterministic algorithm that outputs a $(1+\varepsilon)$-approximation to the minimum cut value of a simple, undirected graph in a single pass over a stream requires $\Omega(n/\varepsilon^2)$ bits of space*

Proof. A randomized insertion-only streaming algorithm for a for-all cut sparsifier needs $\Omega(n/\varepsilon^2)$ bits of space [ACK+16] (since any data structure that stores this information needs that much space). If we carefully look at the proof of [ACK+16] in section 3.1, their hard instance is a disjoint union of $\varepsilon^2 n/2$ bipartite graphs with $2/\varepsilon^2$ vertices each. They use the cut sparsifier to query cuts that have vertices only in one of these disjoint graphs, which implies that the cut sparsifier only needs to preserve the cut values for cuts that have at most $1/\varepsilon^4$ edges. Thus, a $(1+\varepsilon)$ cut sparsifier that preserves the cut values of all cuts of size at most $1/\varepsilon^4$ edges needs $\Omega(n/\varepsilon^2)$ bits of space.

We will now show how to simulate a $(1+\varepsilon)$ cut sparsifier that preserves the cut values of all cuts of size at most $1/\varepsilon^4$ using an algorithm for $(1+\varepsilon)$-approximate minimum cut. This will give us the desired lower bound. Let A be a deterministic insertion-only streaming algorithm for $(1+\varepsilon)$-approximate minimum cut value we run during the stream for input graph G.

After the stream, consider any cut S whose value we want to approximate within a $(1+\varepsilon)$ factor (note that the value is at most $1/\varepsilon^4$). We now add some extra edges to the stream for A. We first construct two cliques C_1 and C_2 on $2n$ vertices each and add their edges to the stream. We then add edges between all vertices of S and C_1 and similarly for all vertices in $\bar{S}$ and C_2. Call this new graph G'. We now end the stream and look at the output of A. We claim this is the $(1+\varepsilon)$-approximate value of cut S in the original graph.

To prove this, we must show that S is the minimum cut in the modified graph G'. We know that the value of S is at most $1/\varepsilon^4 \leqslant n$ (since $\varepsilon \geqslant n^{-1/4}$). Consider any cut that separates vertices in $C_1 \cup S$. Such a cut has a size of at least $2n$. The same applies to any cut separating vertices in $C_2 \cup \bar{S}$. Thus, the minimum cut is $C_1 \cup S$. None of the new edges we added cross this cut, so the size of this cut remains unchanged in G', implying that we get a $(1+\varepsilon)$-approximation to the cut value in G. Note that the algorithm A has to

be deterministic for us to do this because we have to repeat this process for exponentially many cuts (potentially 2^n).

The space taken to build the sparsifier for insertion-only streams is the space taken for A when run on $5n$ vertices. This implies that any deterministic insertion-only streaming algorithm for $(1+\varepsilon)$-approximate minimum cut value needs $\Omega(n/\varepsilon^2)$ bits of space, proving the lower bound for deterministic algorithms. ∎

Proof of Theorem 3. The proof of the theorem directly results from combining Lemma 3.5.1 and Lemma 3.5.2. Notably, this shows that the spectral sparsifier result in Theorem 2 is optimal in space complexity up to polylogarithmic factors. ∎

3.6 Lower Bound: Approximate Minimum Cut

This section gives an alternate proof of the randomized algorithm lower bound for finding an approximate minimum cut (Theorem 3).

Theorem 3. *Fix $\varepsilon > 1/n$. Any randomized algorithm that outputs a $(1+\varepsilon)$-approximation to the minimum cut of a simple, undirected graph in a single pass over a stream with probability at least $2/3$ requires $\Omega(n/\varepsilon)$ bits of space.*

Notably, this shows that the result in Theorem 2 is optimal in space complexity up to polylogarithmic factors. In Section 3.5, we have shown this lower bound for randomized algorithm using the k-edge-connectivity problem in [SW15]. Below we also provide an alternative self-contained construction for the randomized result using the Index communication problem.

Existing work has shown an $\Omega(n^2)$ bits of space lower bound for one-pass streaming algorithms that estimate the exact value of the minimum cut on undirected graphs [Zel11]. This is done via a standard reduction from the Index problem in communication complexity. We use an extension of the same technique to prove a tight lower bound on approximate minimum cut streaming algorithms.

3.6.1 Warm up: Exact Minimum Cut

For the purpose of demonstration, we first give an overview of the proof for the exact minimum cut lower bound given by Theorem 1 of [Zel11]. Recall that it is a standard known result in communication complexity that any randomized protocol solving the following Index problem on a n-bit binary vector with probability at least $2/3$ requires $\Omega(n)$ bits of memory. Suppose that there exists a one-pass streaming algorithm $\mathcal{A}$ which computes

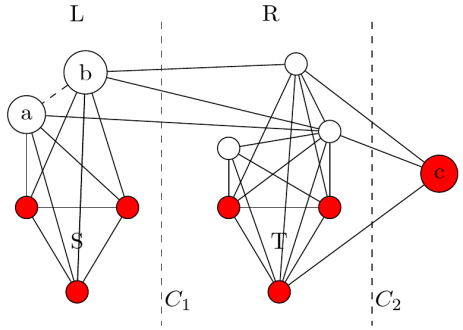


Figure 3.1: Construction of graph H^+ for exact minimum cut from [Zel11]. Red nodes are the ones that are added in addition to the original graph.

the exact minimum cut value of any arbitrary simple unweighted graph of n vertices using $o(n^2)$ bits of space with probability at least $2/3$. The reduction is given as follows. Alice has a bit vector x of length $(n^2 - n)/2$, which she uses to represent the upper half of the adjacency matrix of the graph G. She instantiates $\mathcal{A}$ with input graph H with $7n + 1$ vertices, where the edges of H will be determined later. The edges between the first n nodes are the same as those in G. After feeding the edges of G into $\mathcal{A}$, Alice sends the memory configuration of $\mathcal{A}$ to Bob. Alice also sends the degree of each of the n vertices in G to Bob. In total, this is $O(n \log n) + o(n^2) = o(n^2)$ bits which are sent to Bob.

Bob has an index $i \in [1, (n^2 - n)/2]$, corresponding to an edge that he wishes to determine whether it is in G. We call this potential edge (a, b). Bob then adds edges to H to construct a new graph H^+ under the following rules. He first adds edges into H to construct two disjoint cliques S and T of $3n$ vertices each (here, the nodes in S and T are disjoint with the first n nodes in H). Bob next adds edges from all vertices in S to a and b, and edges from all vertices in T to all vertices in $V \setminus \{a, b\}$. We denote sets $L = S \cup \{a, b\}$ and $R = T \cup V \setminus \{a, b\}$.

Bob then adds edges from the last remaining vertex, which we call c, to an arbitrary set of $\deg_G(a) + \deg_G(b) - 1$ vertices in $V \setminus \{a, b\}$ (see Figure 3.1 for a complete diagram of the construction). He then queries $\mathcal{A}$ about the minimum cut value. The observation is that there are two possible minimum cuts: $C_1 = (L, R \cup \{c\})$ with size $\deg_G(a) + \deg_G(b) - 2$ if ab is an edge in G, or $C_2 = (L \cup R, \{c\})$ with size $\deg_G(a) + \deg_G(b) - 1$ if ab is not an edge in G. From this, Bob then determines the value $x[i]$ with $o(n^2)$ communication with probability at least $2/3$, a contradiction.

3.6.2 Approximate Minimum Cut

We are ready to give proof for Theorem 3. First, we assume that $1/\varepsilon < n$, as otherwise there already exists the known $\Omega(n^2)$ lower bound for minimum cut. Additionally, we assume $\varepsilon = o(1)$, as a constant approximation for minimum cut answers whether a graph is connected or not, and there already exists a $\Omega(n)$ lower bound for connectivity [FKM+08].

Suppose there exists a one-pass streaming algorithm $\mathcal{A}$ which computes a $(1+\varepsilon)$-approximation to the minimum cut of any arbitrary simple unweighted graph on n vertices using $o(n/\varepsilon)$ bits of space and with probability at least $2/3$. Alice has a bit vector x of length $s = \Theta(n/\varepsilon)$, which she uses to represent the edges of a graph G, which is formed by εn disjoint graphs each with $1/(4\varepsilon)$ vertices. She instantiates $\mathcal{A}$ with an input graph H with $(n + 6/\varepsilon)/4 + 1$ vertices, in which the edges between the first n nodes are the same as the edges in G. After feeding the edges of G into $\mathcal{A}$, she then sends the memory configuration of $\mathcal{A}$ to Bob. Alice also sends the degree of each of the $n/4$ vertices in G to Bob. In total this is $O(n\log(1/\varepsilon)) + o(n/\varepsilon) = o(n/\varepsilon)$ bits sent to Bob.

Bob has an index $i \in [1, s]$, which corresponds to an edge within G that he wishes to determine whether it exists or not. Specifically, this edge belongs to one of the εn disjoint subgraphs. Let us denote this subgraph as G_i. Bob now constructs a new graph $H^+(\varepsilon)$ under the following rules. He constructs two cliques S and T, each of size $3/(4\varepsilon)$. For subgraph G_i, Bob connects S, T, and vertex c in the same manner as the exact minimum cut construction. For all other vertices in subgraphs other than G_i, Bob does not follow this construction and instead arbitrarily connects each one of them to all vertices in either S or T. Denote the set of all subgraphs connected to S as G_S, and G_T similarly. See Figure 3.2 for a complete diagram of the construction.

Bob then queries $\mathcal{A}$ to get a $(1+\varepsilon)$-approximation of the minimum cut value of $H^+(\varepsilon)$. We denote sets $L = S \cup \{a, b\}$ and $R = T \cup V_i \setminus \{a, b\}$, where V_i represents the vertices in graph G_i. Once again define the two candidate minimum cuts as $C_1 = (L \cup G_S, R \cup G_T \cup \{c\})$ with size $\deg_G(a) + \deg_G(b) - 2$ if ab is an edge in G, and $C_2 = (L \cup G_S \cup R \cup G_T, \{c\})$ with size $\deg_G(a) + \deg_G(b) - 1$ if ab is not an edge in G. The crucial observation here is that the two candidate minimum cuts are still C_1 and C_2, as shown by the following claim:

Claim 3.6.1. *All graph cuts in $H^+(\varepsilon)$ besides C_1 and C_2 have size at least $3/(4\varepsilon) - 1$.*

Proof. We will prove our claim by describing all cuts with size $< 3/(4\varepsilon) - 1$. First, any cut that separates any two vertices within L (or R, respectively) must have size at least $3/(4\varepsilon) - 1$, from an identical argument used by the construction in [Zel11]. Additionally, any cut that splits any vertex within G_S from S (G_T from T respectively) must also have size at least $3/(4\varepsilon) - 1$, as every vertex in G_S connects to all vertices in S.

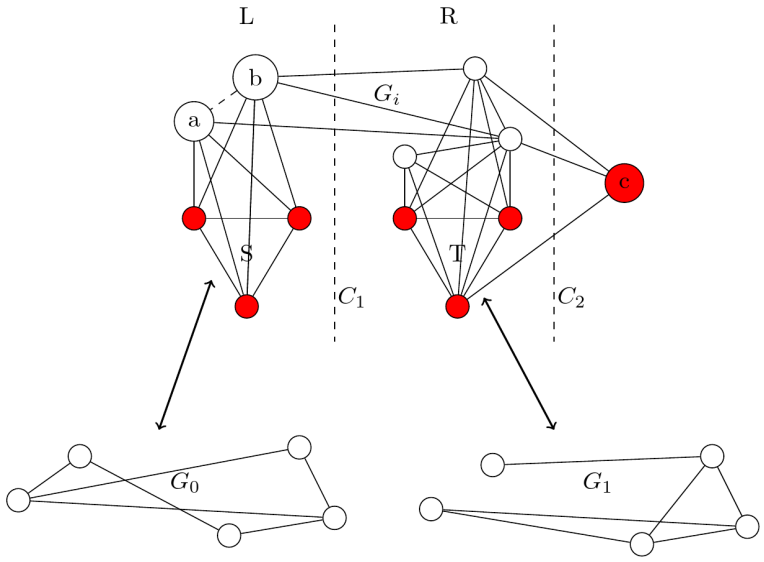


Figure 3.2: Construction of graph $H^+(\varepsilon)$ for approximate minimum cut. Red nodes are the ones that are added in addition to the original graph. G_i represents the graph containing (potential) edge ab, and G_0 and G_1 represent the remaining graphs. The double arrow edges represent bicliques between graph $G_0 \in G_S$ and S, and graph $G_1 \in G_T$ and T.

Therefore, any cut with size $< 3/(4\varepsilon) - 1$ cannot split any part of either $L \cup G_S$ or $R \cup G_T$. We can thus contract both sets into a single super-vertex with parallel edges. From this, the only vertices that remain are c and the super-vertices of $L \cup G_S$ and $R \cup G_T$, and it is easy to verify the only remaining cuts of size $< 3/(4\varepsilon) - 1$ are C_1 and C_2. ∎

Both cuts C_1 and C_2 have a maximum value of $< \deg_G(a) + \deg_G(b) < 1/(2\varepsilon)$. Hence a $(1+\varepsilon)$ approximation gives us the cut value with an additive error strictly less than 0.5, which is enough to distinguish whether ab is in graph G.

3.7 Exact Minimum Cut in Random-Order Streams

In this section, we give an $\widetilde{O}(n)$ algorithm that outputs the minimum cut in a simple, unweighted graph in a single-pass random-order stream. Recall that in a simple graph, there is at most one edge between any pair of vertices. Consider the following algorithm that proves Theorem 4.

High Level Idea. We will use the algorithm in Proposition 3.2.3 (for-all sparsifier in a stream) as a subroutine. The whole algorithm is given in Algorithm 3. Below, we first give

a high-level explanation of our algorithm. We first consider the case when the minimum cut size is $\lambda = \Omega(\log n)$. Since we are now considering the random-order model, a prefix of the edges gives us partial information about the cut sizes. Let H be the subgraph of G which is formed by a prefix of the edges of the stream where $|H| \approx \frac{|G| \log n}{\lambda}$ and $|G|$, $|H|$ are the number of the edges in G and H respectively. By a Chernoff bound, one can show that with high probability, for every subset $S \subset V$, $w_H(S, V \setminus S)$ is a small constant approximation to $w_G(S, V \setminus S)$. H might be too large to store in memory, so we store H_1, a $(1+\varepsilon)$-for-all sparsifier of H, during the stream (we will decide ε later). Therefore, using the graph H_1, we know all cuts in G whose cut size is within a factor of 1.1 of the true minimum cut of G. After this step, when a new edge $e \in G \setminus H$ arrives, we can use graph H_1 to check whether e belongs to some 1.1-approximate minimum cut in G, and if so, we save this edge (from Proposition 3.7.4 we know that the total number of these edges saved is at most $O(n)$). The next question is how to estimate the exact value of these 1.1-approximate cuts in H. The crucial observation is since the minimum cut size of H is an integer and is at most $\Theta(\log n)$, we can set the approximation parameter $\varepsilon = 1/\log^2 n$ for H_1. This gives the exact value of $w_H(S, V \setminus S)$ if S and $V \setminus S$ is a constant-approximate minimum cut in G (since the error is at most $O(\log n) \cdot 1/\log^2 n < 1$). Putting these things together, after the stream, we can enumerate all 1.1-approximate minimum cuts and obtain their exact values, thus getting the exact minimum cut value.

The remaining case to handle is when the minimum cut size is $O(\log n)$. Similarly, in this case, setting $\varepsilon = 1/\log^2 n$ for the cut sparsifier of the full graph will give us the exact value of the minimum cut. Finally, we do not know the value of s, so we will try all powers of λ for it. Note that we do not need the exact value; a 2-approximation to λ will suffice. The detailed description of the algorithm is as follows:

Algorithm 3. Random-order Minimum Cut

Input: Undirected and unweighted graph $G = (V, E)$ in a random-order stream.

Output: The exact minimum cut of G.

Pre-processing:

1. Initialize H_i for $i \in [2 \log n]$, as a cut sparsifier of the first 2^i edges of G with parameter $\varepsilon = 1/\log^2 n$ (Proposition 3.2.3).
2. Initialize empty sets $T_i \leftarrow \emptyset$ for $i \in [2 \log n]$.
3. Initialize degree counters d_i for all nodes $i \in [n]$.

Streaming phase: Let $e_t = (u_t, v_t)$ be the t^{th} edge of the stream.

1. *Update degrees:* $d_{u_t} \leftarrow d_{u_t} + 1,\ d_{v_t} \leftarrow d_{v_t} + 1$.
2. *Update sparsifiers:* For every i with $t \leqslant 2^i$, update H_i with e_t.
3. *Collect witness edges:* For every j with $t > 2^j$:
   (a) If the current edge e_t crosses some non-singleton (1.2)-approximate minimum cut in H_j, add e_t to T_j.
   (b) If $|T_j| > c \cdot n$, stop adding edges to it.

Post-processing:

1. Let $\widetilde{\lambda}$ be the minimum cut of $H_{2\log n}$ and let $p^* := \frac{40\log n}{(0.05)^2\widetilde{\lambda}}$.
2. H^* is the sparsifier over the prefix of size $m \cdot p^*$ edges and T^* is its corresponding witness edge set.
3. For each cut $(S, V \setminus S)$ that is a non-singleton 1.2-approximate minimum cut in H^*, compute:
$$v_S \leftarrow w_{H^*}(S, V \setminus S) + |\{e \in T^* : e \text{ crosses } (S, V \setminus S)\}|$$
4. Output $\min\{\min_S v_S, \min_{i \in V} d_i\}$ as the exact minimum cut value of G along with the corresponding cut.

We now show that the algorithm correctly computes the exact minimum cut of G. We begin by considering a few simple cases. First, if the minimum cut is a singleton cut, then it corresponds to the minimum degree, which the algorithm stores exactly. Thus, we may assume without loss of generality that the minimum cut is non-singleton. We next consider the case when the minimum cut size $\lambda_G < 80\log n/(0.05)^2$. In this case, the cut sparsifier $H_{2\log n}$ (which has parameter $\varepsilon = 1/\log^2 n$) is enough to get the exact size of the minimum cut of G.

Thus, we now solve the problem under the assumptions that the minimum cut is non singleton and has size at least $80\log n/(0.05)^2$. This also ensures that $p^* \leqslant 1$ since $\widetilde{\lambda}$ is a $(1 + 1/\log^2 n)$ approximation to λ. We now consider Karger's uniform sampling lemma.

Proposition 3.7.1 ([Kar94]). *Let $G = (V, E)$ be an unweighted, undirected graph with minimum cut value λ. Sample every edge independently with probability $p \geqslant \frac{20 \log n}{\varepsilon^2 \lambda}$, and let H be the graph on the vertex set V with the sampled edges having weight $1/p$. Then H is a $(1+\varepsilon)$-cut sparsifier of G with high probability.*

Notice that a prefix of the random-order stream with $m \cdot p$ edges is similar to H in Proposition 3.7.1 with a key difference. The number of edges in the prefix is fixed ($m \cdot p$) so the sampling indicator random variables are negatively associated. Fortunately, the same version of Chernoff bounds also work for negatively associated random variables Proposition 2.4.3. As a result, the proof of Proposition 3.7.1 remains valid with only a minor modification—replacing independence with negative association. [Waj17, Ber24] show that such fixed-size samples (i.e., random-order stream prefixes) exhibit negative association, allowing the concentration bounds to carry over directly. We now prove the following claim:

Claim 3.7.2. *H^* with weights increased by a factor of $1/p^*$ is a 1.1-approximate cut sparsifier of G with high probability.*

Proof. We can apply the Proposition 3.7.1 with $\varepsilon = 0.05$ and conclude that if we construct H by sampling at least $20m(\log n)/(0.05)^2\lambda \leqslant m \cdot p^*$ edges of G then H with weights $1/p^*$ is a $(1 + 0.05)$ cut sparsifier of G. The inequality holds because $\widetilde{\lambda}$ is a $(1 + 1/\log^2 n)$ approximation to λ. Note that in our case, these are the first $m \cdot p^*$ edges of the random-order stream. We have that H^* is the cut sparsifier of H with parameter $1/\log^2 n$. Therefore, H^* with weights increased by a factor of $1/p^*$ is a 1.1 cut sparsifier of G (since H with weights increased by a factor of $1/p^*$ is a sparsifier of G with parameter 0.05). ∎

We now show that all the edges stored in T^* are edges that belong to 1.4-approximate minimum cuts of G.

Claim 3.7.3. *All edges stored in T^* belong to 1.4-approximate minimum cuts of G.*

Proof. The only edges that are stored in T^* are edges that belong to 1.2-approximate minimum cuts of H^*. We know that H^* with weights increased is a 1.1-approximate minimum cut of G (Claim 3.7.2). Also, reweighting does not change what the approximate minimum cuts are. Thus, these cuts are at most $1.1 \cdot 1.2 \leqslant 1.4$ approximate cuts of G. ∎

We also need the following structural result for the approximate minimum cuts of a graph.

Proposition 3.7.4 ([RSW18]). *For any simple, unweighted graph and any constant $\varepsilon > 0$, the total number of edges that participate in non-singleton $(2-\varepsilon)$-approximate minimum cuts is at most $O(n)$.*

This shows that the number of edges that belong to all 1.4-approximate minimum cuts is at most $c \cdot n$ for some constant c. Thus, T^* has at most $c \cdot n$ edges (Claim 3.7.3) and so we never stop adding edges to it in Item 3b of Algorithm 3.

Let G^* be the graph created by the first $m \cdot p^*$ elements of the stream (so H^* is a cut sparsifier of G^*). We now show that the number of edges of constant approximate minimum cuts that are sampled in G^* is $\Theta(\log n)$.

Claim 3.7.5. *For every 1.4-approximate minimum cut of G, the number of edges crossing it that are sampled in G^* is $\Theta(\log n)$ with high probability.*

Proof. Consider any 1.4-approximate minimum cut S of G. G^* contains $m \cdot p^*$ uniformly random edges of G. Thus, the probability that an edge is sampled in G^* is exactly p^*. Let X_i be the random variable that is 1 if the i^{th} edge crossing the cut is sampled in G^* and 0 otherwise. Let $X = \sum_i X_i$ be the number of edges crossing the cut that are sampled.

In expectation the number of sampled edges will be:

$$\mathbb{E}[X] = \sum_i \mathbb{E}[X_i] = \lambda_G(S) \cdot (40 \log n)/(0.05)^2 \widetilde{\lambda} = \Theta(\log n).$$

The last equality holds because $\lambda_G(S)$ and $\widetilde{\lambda}$ are both 1.4 factor approximations of λ_G. We can now show concentration using the Chernoff bound for negatively associated random variables (Proposition 2.4.3). Since X_i's are negatively associated and in $[0,1]$ we have:

$$\Pr(|X - \mathbb{E}[X]| > \mathbb{E}[X]/2) \leqslant 2\exp(\mathbb{E}[X]/12) \leqslant 1/n^{10}.$$

The last inequality holds by plugging in the value of p^* and noting that $\lambda_G(S)$ and $\widetilde{\lambda}$ are both 1.4 factor approximations of λ_G. Union bounding over the $O(n^3)$ 1.4-approximate minimum cuts concludes the proof. ∎

We now show that all edges of small approximate minimum cut of G are preserved.

Claim 3.7.6. *Let S be a non-singleton 1.05-approximate minimum cut of G. All edges crossing S are preserved by Algorithm 3 with high probability.*

Proof. Consider a 1.05-approximate minimum cut S of G. We know from Claim 3.7.5 that the number of edges crossing S sampled in G^* is $\Theta(\log n)$ with high probability. H^* is a cut sparsifier of G^* with parameter $1/\log^2 n$ which implies that the error in the size of cut S can be at most $o(1)$. Thus, H^* preserves all the edges that cross the cut S.

We also know that T^* contains all the edges in the suffix (after the first $m \cdot p^*$ edges) that belong to 1.2-approximate minimum cuts of H^*. In H^* this corresponds to cuts of size at most $1.2 \cdot \lambda_{H^*}$. If we increase the weights of H^* by a factor of $1/p^*$, this corresponds to cuts that have sizes between $(1/p^*) \cdot \lambda_{H^*}$ and $(1/p^*) \cdot 1.2 \cdot \lambda_{H^*}$.

In the worst case, $(1/p^*) \cdot \lambda_{H^*}$ can be $(0.9) \cdot \lambda_G$ but cannot be smaller because H^* with weights increased by a factor of $1/p^*$ is a 1.1-approximate cut sparsifier of G with high probability (Claim 3.7.2). Thus, we have that S has weight at most $(1.05) \cdot \lambda_G < (1.2) \cdot (0.9) \cdot \lambda_G$. Therefore, all edges of S will be preserved in the prefix and suffix. ∎

We are now ready to prove Theorem 4.

Proof of Theorem 4. We have shown that all 1.05-approximate minimum cuts of G are preserved along with all edges with high probability. In Algorithm 3, the list of cuts we take the minimum over are the 1.2-approximate minimum cuts of H^*. For these cuts, all the edges crossing this cut in G are preserved. The proof of this is similar to what we did above. Firstly, these 1.2-approximate minimum cuts of H^* are 1.4-approximate minimum cuts are G, and their sizes are $\Theta(\log n)$ in G^* (Claim 3.7.5). This implies that H^* will preserve all their edges in the prefix because it has parameter $1/\log^2 n$. Also, we explicitly store all their edges in the suffix.

Thus, for non-singleton cuts, we are taking a minimum over a list of cuts all of which have all edges of G preserved. Also, this list includes all the 1.05-approximate minimum cuts of G and in particular the non-singleton minimum cut of G. Finally, comparing with the minimum degree, we get the exact minimum cut of G.

We now show that the space complexity is $\widetilde{O}(n)$. We run $O(\log n)$ instances of the cut sparsifier algorithm in Proposition 3.2.3 with $\varepsilon = 1/\log^2 n$, which takes space $\widetilde{O}(n)$. We then store at most $c \cdot n$ edges for each of the $O(\log n)$ copies. We also maintain the degree of each node during the stream, which takes $\widetilde{O}(n)$ bits of space. Putting everything together, the space usage of our algorithm is $\widetilde{O}(n)$ bits. ∎

We also note that our algorithm not only identifies all exact minimum cuts of G along with the minimum cut value, but also preserves all edges participating in small approximate minimum cuts of G.

Chapter 4

Streaming Vertex Connectivity

4.1 Introduction

The *vertex connectivity* of an undirected graph $G = (V, E)$, with n vertices and m edges, is the size of the smallest vertex cut in G, defined as the minimum number of vertices whose removal disconnects the graph (or turns it into a singleton vertex). Finding the vertex connectivity of a graph is a fundamental problem in combinatorial optimization and has been extensively studied in the literature; see, e.g., [Kle69, Pod73, ET75, BDD+82, LLW88, HRG00, LNP+21]. This problem can be solved in "polylogarithmic max-flow time" via a result of [LNP+21], which combined with the recent breakthrough improvement for max-flow computation in [CKL+22], leads to an $m^{1+o(1)}$ time algorithm for vertex connectivity.

We study the vertex connectivity problem in *dynamic streams*. In this model, the edges of the input graph G are presented to the algorithm as a sequence of both edge insertions and deletions. The goal is to, given an integer k at the start of the stream, process the stream with limited space and at the end output whether or not the graph is *k-vertex-connected*, namely, its vertex connectivity is at least k. In fact, in our algorithm, we focus on not only deciding if the graph is k-vertex connected or not, but also outputting a *certificate* of k-vertex-connectivity defined as follows:

Definition 4.1.1. *For any graph $G = (V, E)$, a* **certificate** *of k-vertex-connectivity for G is a subgraph on the same vertex set $H = (V, E_H)$ such that G is k-vertex-connected if and only if H is k-vertex-connected.*

A certificate of k-vertex-connectivity needs $\Omega(kn)$ edges since any k-vertex-connected graph has at least $kn/2$ edges, as it needs to have a minimum degree of at least k. Mader's

theorem (see Proposition A.1.2) implies that there is also always a certificate with $O(kn)$ edges although we will not use this result directly in our dynamic streaming algorithm.

In insertion-only streams with no edge deletions, it has been known since the introduction of the model in [FKM+04], that one can find a certificate of k-vertex connectivity in $\widetilde{O}(kn)$ space using the sparsification techniques of [CKT93] or [EGIN97][1], which is nearly optimal. Moreover, [SW15] proved that $\Omega(kn)$ space is needed even for the original (decision) problem, thus settling the space complexity of the problem in insertion-only streams, up to logarithmic factors. But when it comes to dynamic streams, the best upper bound achieves $\widetilde{O}(k^2n)$ space [GMT15] with no better known lower bounds. We close this gap in this paper.

Our main result is a general (model-independent) approach for computing a vertex connectivity certificate.

Result 4.1 (Formalized in Theorem 6)**.** *For any graph G and integer $k \geqslant 1$, let H be a subgraph of G with $O(kn \log n)$ edges obtained as a union of $O(k^2 \log n)$ spanning forests, each on a random subset of $O(n/k)$ vertices chosen independently of the others. Then, with high probability, H is a certificate of k-vertex-connectivity for G. This certificate also preserves all vertex cuts of size up to k and can determine if any two given vertices are k-vertex-connected or not.*

Result 4.1 reduces the problem of finding a certificate of k-vertex-connectivity to computing spanning forests on random subsets of vertices. This gives a general approach for solving k-vertex-connectivity across different models that can be of its own independent interest. In particular, Result 4.1 immediately leads to a dynamic streaming algorithms for k-vertex-connectivity when combined with the dynamic streaming algorithms of [AGM12a] for computing spanning forests.

Result 4.2 (Formalized in Theorem 7)**.** *There is a randomized single-pass $\widetilde{O}(kn)$ space algorithm that solves k-vertex-connectivity with high probability in dynamic streams.*

Result 4.2 also closes one of the last remaining gaps in understanding of dynamic versus insertion-only streams. Starting with the breakthrough of [AGM12a] that initiated the study of dynamic graph streams, various graph problems such as cut sparsifiers [AGM12b], spectral sparsifiers [KLM+17], densest subgraph [MTVV15], subgraph counting [AGM12b],

[1]While this connection has been observed in several places, e.g., in [SW15, GMT15], we are not aware of a concrete reference that includes this proof in the streaming model and thus provide a self-contained proof in Section A.1 for completeness.

and $(\Delta+1)$-vertex coloring [ACK19a] were shown to admit algorithms in dynamic streams with similar guarantees as those of insertion-only streams. In particular, for the closely related problem of k-edge-connectivity, it was shown already by [AGM12a] how to obtain similar bounds in dynamic streams as insertion-only streams. For a few other problems such as maximum matchings and minimum vertex cover, strong separations between the two models were proven in [Kon15, AKLY16] (see also [DK20, AS22, NS22]); a conjectured separation for the shortest path problem also appears in [FKN21]. Before our Result 4.2 however, it was not clear vertex connectivity belongs to which family of these problems.

As a secondary result, we also extend the previous $\Omega(kn)$ lower bound of [SW15] to multiple passes.

Result 4.3 (Formalized in Theorem 8)**.** *Any randomized p-pass streaming algorithm that solves k-vertex-connectivity with constant probability even in insertion-only streams, needs $\Omega(kn/p)$ space.*

Result 4.3 is proven by lower bounding the communication complexity of k-vertex-connectivity problem with $\Omega(kn)$ bits. This also answers an open problem of [BBE+22] in negative that asks whether recent equivalences between vertex connectivity and vertex-capacitated max-flow in [LNP+21] in classical setting also extends to communication complexity model (a recent result of [BBE+22] shows that vertex-capacitated max-flow (unit capacity) can be solved with $\widetilde{O}(n)$ communication, while our communication lower bound in general implies an $\Omega(n^2)$ lower bound for determining the vertex connectivity of a graph, thus ruling out such an equivalence). We remark that our Result 4.3, similar to the previous lower bound for streaming vertex connectivity in [SW15], holds on multi-graphs (with at most two parallel edges per pairs of vertices); our algorithm in Result 4.2 also works for multi-graphs. It remains an interesting open question to prove any streaming lower bound for vertex connectivity on simple graphs as well (even for single-pass algorithms).

Our techniques. Result 4.1 (and by extension Result 4.2) is obtained via a novel and improved *analysis* of the dynamic streaming algorithm of [GMT15] (with minor modifications), that leads to an improved bound on the size of the certificate[2]. [GMT15] presented algorithms for two *relaxations* of the vertex connectivity problem in dynamic streams:

- k-vertex-**query**-connectivity problem: the algorithm is additionally given a set X of size at most k *after* the stream and the goal is to determine whether removing

[2]The algorithm in [GMT15] is presented as a streaming algorithm but it implicitly gives a certificate with $O(k^2 n \log n)$ edges.

X from G disconnects the graph or not (this is similar to the vertex connectivity oracle problems studied in [LS22, PSY22]). [GMT15] showed that this problem can be solved in $\widetilde{O}(kn)$ space (and that this is also nearly optimal for this problem).

- **promised-gap** k-vertex-connectivity problem: the algorithm is given a parameter $\varepsilon \in (0,1)$ at the *start* of the stream and the goal is to determine whether the vertex connectivity of the input graph G is at most k or at least $(1+\varepsilon)\cdot k$. [GMT15] showed that this problem could be solved in $\widetilde{O}(kn/\varepsilon)$ space (using an algorithm quite similar to the one for the previous case).

Both these algorithms work roughly as follows (think of $\varepsilon = \Theta(1)$ for the second one in this context): for $\widetilde{O}(k^2)$ times in parallel, sample $\widetilde{O}(n/k)$ vertices from the input graph uniformly at random and maintain a spanning forest on these vertices using the dynamic streaming algorithm of [AGM12a] (this is basically the same approach taken in our Result 4.1). Then, solve the problem on these stored set of edges at the end of the stream. This approach can be implemented in $\widetilde{O}(k^2)\cdot\widetilde{O}(n/k) = \widetilde{O}(kn)$ space and [GMT15] proves, using a somewhat different analysis, that this solves the problem in each case with high probability.

One can solve the original k-vertex-connectivity problem using this approach as follows. For the query problem, boost the probability of success of the algorithm to $1-n^{-k}$ by running the algorithm in parallel $\Theta(k\log n)$ times; this allows for taking a union bound over at most $\binom{n}{k}$ possible choices for the query set X and testing whether removal of *any* of them can disconnect the graph. For the promised-gap problem, we can set $\varepsilon = 1/k$ which allows us to distinguish between graphs with vertex connectivity k versus $k-1$ and thus solve the k-vertex-connectivity problem. Nevertheless, as is apparent, either of these solutions leads to an algorithm with $\widetilde{O}(k^2 n)$ space which is sub-optimal.

The key novelty in our work is another analysis of essentially the same vertex-sampling plus spanning forest computation approach of [GMT15]. This analysis allows us to "beat the union bound" over all possible k-subsets of vertices as candidate choices for the vertex cut, that was the source of the additional factor k in space in the algorithm of [GMT15]. In particular, our analysis consists of two parts. We first show that all pairs of vertices with sufficiently "high" vertex connectivity, say, at least $2k$, remain at least k-vertex-connected even over the stored edges of the sampling approach (this part is quite similar to the guarantee of promised-gap algorithm of [GMT15]). We then prove that *all* edges in the input graph with "low" vertex connectivity between their endpoints, say, less than $2k$, are recovered by this sampling approach entirely[3]. Finally, we combine these two parts to

[3]The fact that the number of these edges itself is sufficiently small is a direct corollary of Mader's

argue that the sampled set of edges is a certificate for k-vertex-connectivity of the input graph and conclude the proof. This proof more generally shows that *all* minimum (global) vertex cuts as well as all s-t vertex cuts of size up to k are preserved in this sampling process.

Preliminaries. We use the algorithm of [AGM12a] that can find a spanning forest of a graph in a dynamic stream.

Proposition 4.1.2 ([AGM12a])**.** *There is an algorithm that given any N-vertex graph G in a dynamic stream and $\delta \in (0,1)$, computes a spanning forest T of G with probability at least $1-\delta$ in $O(N \log^3(N/\delta))$ space.*

4.2 A Certificate of Vertex Connectivity

We present a certificate of k-vertex-connectivity in this section, formalizing Result 4.1. Our algorithm is virtually identical (up to changing constants and ignoring implementation details in dynamic streams) to the algorithm in [GMT15] for the k-vertex-**query**-connectivity problem mentioned in the introduction. However, we provide an improved analysis showing that it also works for the k-vertex-connectivity problem.

Algorithm 4. An algorithm for computing a certificate of k-vertex-connectivity.

Input: A graph $G = (V, E)$ and an integer k.

Output: A certificate H for k-vertex-connectivity of G.

1. For $i = 1, 2, \ldots, r := 200k^2 \ln n$, do the following:
   - Let V_i be a subset of V where each vertex is sampled independently with probability $1/k$.
   - Let $G_i = G[V_i]$ be the induced subgraph of G on V_i.
   - Compute a spanning forest T_i of G_i.
2. Output $H := T_1 \cup T_2 \cup \ldots \cup T_r$ as a certificate for k-vertex-connectivity of G.

theorem (Proposition A.1.2) that states that every graph with $O(kn)$ edges has a $(2k)$-vertex-connected subgraph. This theorem is at the heart of existing (near) optimal algorithms for vertex connectivity in insertion-only streams (see Section A.1). Nevertheless, since our proof requires additionally recovering these edges via a particular sampling method, it does *not* rely on Mader's theorem, and instead, as a corollary, implies a weaker variant of Mader's theorem (with $O(kn \log n)$ edges instead) via a probabilistic argument quite different from the standard proofs of this theorem (Section A.2).

The following theorem proves the main guarantee of this algorithm.

Theorem 6. *Algorithm 4, given any graph $G = (V, E)$ and any integer $k \geqslant 1$, outputs a certificate H of k-vertex-connectivity of G with $O(kn \cdot \log n)$ edges with high probability.*

The analysis in the proof of Theorem 6 is twofold. We first show that pairs of vertices that are at least $2k$-vertex-connected in G stay k-vertex-connected in H. Secondly, we show that edges whose endpoints are not $2k$-vertex-connected in G will be preserved in H. Putting these together, we then show that H is a certificate for k-vertex-connectivity of G and has at most $\widetilde{O}(kn)$ edges.

We start by bounding the number of edges of the certificate H. We first show that the sum of sizes of V_i is $O(kn \log n)$ with high probability.

Claim 4.2.1. $\sum_{i=1}^{r} |V_i| = O(kn \cdot \log n)$ *with high probability.*

Proof. For any iteration $i \in [r]$, the graph G_i has n/k vertices in expectation. We have $r = O(k^2 \ln n)$ iterations so $\sum_{i=1}^{r} |V_i| = O(kn \cdot \log n)$ in expectation. We prove this is the case with high probability as well.

For $i \in [r]$, let X_i be the random variable denoting the number of vertices in V_i. We know that $0 \leqslant X_i \leqslant n$ (the inequalities are tight when $V_i = \emptyset$ and $V_i = V$). Let $X = \sum_i X_i$ be the random variable governing the sum of sizes of V_i's. We have $\mathbb{E}[X_i] = n/k$ implying $\mathbb{E}[X] = \mu = r \cdot (n/k)$. Using a Chernoff bound (Proposition 2.4.3) with parameters $b = n$ and $\varepsilon = 1$ we get:

$$\Pr(X > 2\mu) \leqslant \exp\left(\frac{-\mu}{4b}\right) = \exp\left(\frac{-r \cdot (n/k)}{4n}\right) = \exp(-50k \ln n) \leqslant n^{-50}.$$

Thus, we get that the $\sum_{i=1}^{r} |V_i| \leqslant 2\mu_H = O(kn \cdot \log n)$ with high probability as well. ∎

Lemma 4.2.2. *The certificate H in Algorithm 4 has $O(kn \cdot \log n)$ edges with high probability.*

Proof. Each spanning forest T_i has at most $|V_i|$ edges. Thus, the total number of edges in H can be bounded by $\sum_{i=1}^{r} |V_i| = O(kn \cdot \log n)$ with high probability (by Claim 4.2.1). ∎

We now prove the correctness of this algorithm in the following lemma.

Lemma 4.2.3. *Subgraph H of Algorithm 4 is a certificate of k-vertex-connectivity for G with high probability.*

Lemma 4.2.3 will be proven in two steps. We first show that every pair of vertices that have at least $2k$ vertex-disjoint paths between them in G have at least k vertex-disjoint paths in H with high probability[4].

Lemma 4.2.4. *Every pair of vertices s, t in G that have at least $2k$ vertex-disjoint paths between them in G have at least k vertex-disjoint paths in H with high probability.*

We then show that every edge whose endpoints have less than $2k$ vertex-disjoint paths between them in G will belong to H as well.

Lemma 4.2.5. *Every edge $(s, t) \in G$ that has less than $2k$ vertex-disjoint paths between its endpoints in G belongs to H also with high probability.*

The proofs of these lemmas appear in the next two subsections. We first use these lemmas to prove Lemma 4.2.3 and conclude the proof of Theorem 6.

Proof of Lemma 4.2.3. We first condition on the events in Lemma 4.2.4 and Lemma 4.2.5 both of which happen with high probability. We also condition on the event that H has the same set of vertices as G. This event also happens with high probability because the probability that a given vertex is not in H is $(1 - 1/k)^r \leqslant \exp(-200k \cdot \ln n) = n^{-200k}$ and hence by a union bound, all vertices in G are also in H with high probability. All in all, by a union bound, all the above events happen together with high probability.

We need to show that H is k-vertex-connected iff G is k-vertex-connected. If H is k-vertex-connected then G is also k-vertex-connected simply because H is a subgraph of G (and crucially on the same set of vertices, namely, it is a spanning subgraph).

We now assume towards a contradiction that G is k-vertex-connected, but H is not. This means that there is a vertex cut X of size at most $k - 1$ such that there is a partition (S, X, T) of V with no edges between S and T in H (so that removing X disconnects H). Since G is k-vertex-connected, X cannot be a vertex cut in G and thus G has an edge $e = (s, t)$ between S and T (see Figure 4.1).

We now consider two cases.

- Case 1: s and t have at least $2k$ vertex-disjoint paths between them in G.
  By conditioning on the event of Lemma 4.2.4, we can say that s and t have at least

[4]By Menger's theorem (Proposition A.1.1), this is equivalent to saying any pair of vertices that are $(2k)$-vertex-connected in G remain at least k-vertex-connected in H. However, we do not need to explicitly use Menger's theorem in our proofs.

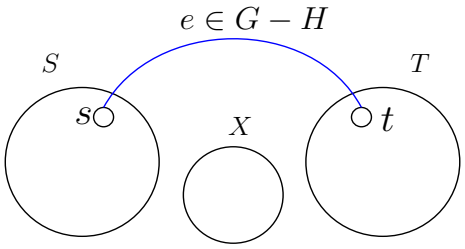


Figure 4.1: An illustration of the partition (S, X, T) of vertices of G and H. There are no edges between S and T in H, while G has at least one edge $e = (s, t)$ between S and T, to ensure its k-vertex-connectivity as $|X| < k$.

k vertex-disjoint paths in H. Deleting X can remove at most $|X| \leqslant k - 1$ of these paths in $H - X$. This implies that there is still an s-t path in $H - X$ and thus there is an edge between S and T in $H - X$, a contradiction.

- Case 2: s and t have less than $2k$ vertex-disjoint paths between them in G.
  Since there are fewer than $2k$ vertex-disjoint paths between s and t in G, by conditioning on the event of Lemma 4.2.5, e would be preserved in H, a contradiction with H having no edge between S and T.

In conclusion, we get that H is a certificate of k-vertex-connectivity for G with high probability. ∎

Theorem 6 now follows immediately from Lemma 4.2.2 and Lemma 4.2.3.

Before moving on from this section, we present the following corollary of Algorithm 4 that allows for using this algorithm for some other related problems in dynamic streams as well.

Corollary 4.2.6. *The subgraph H output by Algorithm 4 with high probability satisfies the following guarantees:*

(*i*) *For any pair of vertices s, t in G, there are at least k vertex-disjoint s-t paths in G iff there at least k vertex-disjoint s-t paths in H (this holds even if G is not k-vertex-connected).*

(*ii*) *Every vertex cut of H with size less than k is a vertex cut in G and vice versa (this means all vertex cuts of G are preserved in H as long as their size is less than k).*

The proof of this corollary is identical to that of Lemma 4.2.3 and is thus omitted.

4.2.1 Proof of Lemma 4.2.4

We prove Lemma 4.2.4 in this part following the same approach as in [GMT15]. For this proof, without loss of generality, we can assume that $k > 1$: for $k = 1$, each graph G_i is the same as G and thus the any spanning forest contains an s-t path which will be added to H, trivially implying the proof.

Fix any pair of vertices s, t with at least $2k$ vertex-disjoint paths between them. We choose an arbitrary set X of vertices with size $k-1$, and the goal is to show that s and t remain connected in the graph $H - X$ with very high probability. We do so by showing that out of the at least k vertex-disjoint paths between s and t in $G - X$, with probability $1 - n^{-\Theta(k)}$, at least one of them is entirely sampled as part of the *subset* of G_i's for $i \in [r]$ that do not contain any vertex from X. This will be sufficient to prove existence of a s-t path in $H - X$. A union bound over the $\binom{n}{k-1}$ choices of X and $\binom{n}{2}$ pairs s, t concludes the proof.

Fix X as a set of $k-1$ vertices that contains neither s nor t. Define:

$$I(X) := \{i \in [r] : V_i \cap X = \emptyset\}; \tag{4.1}$$

that is, the indices of sampled graphs in $G_1, \ldots, G_r$ that contain no vertex from X. We first argue that $|I(X)|$ is large with high probability.

Claim 4.2.7. $\Pr(|I(X)| \leqslant r/8) \leqslant n^{-5k}$.

Proof. Fix any index $i \in [r]$ and a vertex $v \in X$. The probability that v is not sampled in V_i is $(1 - 1/k)$ by definition and thus,

$$\Pr(V_i \cap X = \emptyset) = (1 - 1/k)^{k-1} \geqslant 1/4,$$

given that $k > 1$ (as argued earlier) and the choice of vertices is independent in V_i. Therefore, we have,

$$\mathbb{E}\,|I(X)| = r \cdot (1 - 1/k)^{k-1} \geqslant r/4.$$

By an application of the Chernoff bound (Proposition 2.4.3) with $\mu_L = r/4$ and $\varepsilon = 1/2$, we have,

$$\Pr(|I(X)| \leqslant r/8) \leqslant \exp(-r/4 \cdot 1/10) < n^{-5k}. \qquad \blacksquare$$

In the rest of the proof we condition on the event that $|I(X)| \geqslant r/8$. To continue, we need some definitions. There are more than k vertex-disjoint paths between s and t in $G - X$ since there were $2k$ of them in G and only $k-1$ vertices (set X) are deleted.

Choose k of them arbitrarily denoted by $P_1(X), \ldots, P_k(X)$. For each path $P_j(X)$, let a_j be the edge incident to s, B_j be the remaining path until the final edge c_j which is incident to t – it is possible for a_i and c_i to be the same and B_i be empty (see Figure 4.2 for an illustration).

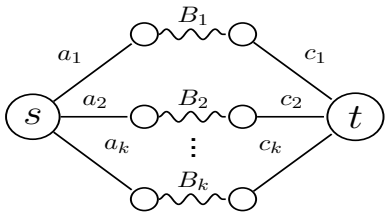


Figure 4.2: An illustration of the s-t paths $P_1(X), P_2(X), \ldots, P_k(X)$. Each $P_j(X)$ consists of an edge a_j from s, a path B_j until the last edge c_j to t.

We define the notion of "preserving" a path.

Definition 4.2.8. *Let $G_X := \cup_{i \in I(X)} G_i$ be the union of graphs indexed in Eq (4.1). We say that a path P in $G - X$ is* **preserved** *in G_X iff for every edge $e \in P$, there exists at least one $i \in I(X)$ such that $e \in G_i$; in other words, the entire path P belongs to G_X.*

We are going to show that with high probability, at least one path $P_j(X)$ for $j \in [k]$ is preserved by G_X. Before that, we have the following claim that allows us to use this property to conclude the proof.

Claim 4.2.9. *If any s-t path $P_j(X)$ for $j \in [k]$ is preserved in G_X then s and t are connected in $H - X$.*

Proof. Given that $P := P_j(X)$ is preserved, we have that for any edge $e = (u, v) \in P$, there is some graph G_i for $i \in I(X)$ that contains e. This means that u, v are connected in G_i which in turn implies that the spanning forest T_i of G_i contains a path between u and v. Moreover, since $i \in I(X)$, we know that G_i and hence T_i contain no vertices of X and thus u and v are connected in $T_i - X$ as well. Stitching together these u-v paths for every edge $(u, v) \in P$ then gives us a walk between s and t in $H - X$, implying that s and t are connected in $H - X$. ∎

We will now prove that some path $P_j(X)$ for $j \in [k]$ is preserved with very high probability.

Claim 4.2.10. *Conditioned on $|I(X)| \geqslant r/8$, the followings three probabilities are each at most n^{-2k}:*

1. $\Pr(a_j \notin G_X$ *for at least* $k/3$ *values of* $j \in [k])$;

2. $\Pr(B_j \not\subseteq G_X$ *for at least* $k/3$ *values of* $j \in [k])$;

3. $\Pr(c_j \notin G_X$ *for at least* $k/3$ *values of* $j \in [k])$.

Proof. To start the proof, note that even conditioned on a choice of $I(X)$, the vertices in each path $P_j(X)$ appear independently in each graph G_i for $i \in I(X)$. This is because these paths do not intersect with X and by the independence in sampling of each graph G_i for $i \in [r]$. Moreover, given that these paths are vertex-disjoint (although share s and t), the choices of their *inner* vertices across each graph G_i for $i \in [r]$, are independent. We crucially use these properties in this proof.

An edge is present in G_i if both of its endpoints are sampled which happens with probability $1/k^2$. Thus, each edge in B_j is not present in G_i with probability $(1 - 1/k^2)$ and hence is not present in G_X with probability $(1 - 1/k^2)^{|I(X)|}$. Hence, by the union bound,

$$\Pr(B_j \not\subseteq G_X) \leqslant |B_j| \cdot \big(1 - 1/k^2\big)^{|I(X)|} \leqslant n \cdot \big(1 - 1/k^2\big)^{r/8} \leqslant n \cdot \exp\big(-200k^2 \ln n / 8k^2\big) = n^{-24}.$$

Finally, note that since the paths B_j for $j \in [k]$ are vertex-disjoint, the probability of the above event is independent for each one. Thus,

$$\Pr(B_j \not\subseteq G_X \text{ for at least } k/3 \text{ values of } j \in [k]) \leqslant \binom{k}{k/3} \big(n^{-24}\big)^{k/3} \leqslant 2^k \cdot n^{-8k} \leqslant n^{-7k}.$$

As such, the entire path B_j will lie inside G_X for at least $2k/3$ values of $j \in [k]$ with very high probability.

The analysis for a_j's and c_j's is slightly different since one of their endpoints, namely, s and t, respectively, is shared across all of them. But the proofs for a_j's and c_j's are entirely symmetric, so we just consider a_j's. Consider the set of indices

$$I_s(X) := I(X) \ \cap \ \{i \in [r] : s \in G_i\}\,;$$

that is the graphs in $I(X)$ which additionally contain the vertex s. For $i \in I_s(X)$, the graph G_i contains the vertex s and but no vertex from X. We know that $\mathbb{E}\,|I_s(X)| = |I(X)|\,/k$ since probability of sampling vertex s in any G_i is $1/k$. By an application of the Chernoff bound (Proposition 2.4.3) with $\varepsilon = 0.5$, we have,

$$\Pr(|I_s(X)| \leqslant |I(X)|\,/2k) \leqslant \exp(-\,|I(X)|\,/10k) \leqslant \exp(-200k^2 \ln n / 80k) = n^{-2.5k}.$$

Moreover, for any $i \in I_s(X)$, the probability that a_j is in G_i is $1/k$. Thus, for any fixed $j \in [k]$,

$$\Pr(a_j \notin G_X) = (1 - 1/k)^{|I_s(X)|}.$$

Combining the above two equations, we have,

$$\begin{aligned}
&\Pr(a_j \notin G_X \text{ for at least } k/3 \text{ values of } j \in [k]) \\
&\leqslant \Pr\Big(|I_s(X)| \leqslant |I(X)|/2k\Big) \\
&+ \Pr\Big(a_j \notin G_X \text{ for at least } k/3 \text{ values of } j \mid |I_s(X)| > |I(X)|/2k\Big) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by the law of total probability)} \\
&\leqslant n^{-2.5k} + \binom{k}{k/3}\left((1-1/k)^{|I(X)|/2k}\right)^{k/3} \\
&\leqslant n^{-2.5k} + 2^k \cdot \left(\exp\left(-200k^2 \ln n/16k^2\right)\right)^{k/3} < n^{-2k}.
\end{aligned}$$

The same property also holds for c_j's by symmetry, concluding the proof. ∎

By union bound over the events of Claims 4.2.7 and 4.2.10, we have that there exists an index $j \in [k]$ such that the path $P_j(X)$ is preserved in G_X. Thus, by Claim 4.2.9, for a fixed choice of X, and s, t, the probability that s and t are not connected in $H - X$ is at most $4n^{-2k}$. A union bound over the choices of X and s, t, then implies that the probability that even one such choice of X and s, t exists is at most

$$\binom{n}{k-1} \cdot \binom{n}{2} \cdot 4n^{-2k} \leqslant n^{k+1} \cdot 4n^{-2k} = 4n^{-k+1} < 4n^{-1},$$

since $k \geqslant 2$. This completes the proof of Lemma 4.2.4.

4.2.2 Proof of Lemma 4.2.5

We now prove Lemma 4.2.5. For this proof also, without loss of generality, we can assume that $k > 1$: for $k = 1$, each graph G_i is the same as G and thus any spanning forest contains the only s-t path, namely, the edge (s, t) (as s and t can only be 1-connected through the edge (s, t)) which will be added to H, thus trivially implying the proof. We now consider the main case.

Fix any pair of vertices $s, t \in G$ which have less than $2k$ vertex-disjoint paths between them. We know that deleting the edge (s, t) and some set of vertices X of size less than $2k$

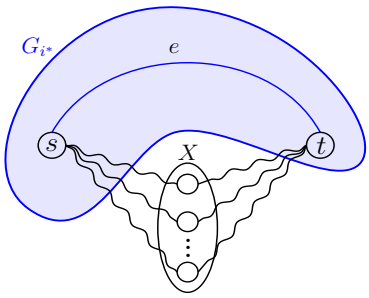


Figure 4.3: An illustration of a good graph G_{i^*} wherein both vertices s and t are sampled and all the vertices in set X are not. Thus, none of the s-t paths, except for the edge e, exist in G_{i^*} since they all pass through X. Therefore, the spanning forest T_{i^*} necessarily contains the edge $e = (s, t)$.

should disconnect s and t. For any $i \in [r]$, we call the graph G_i **good** if it samples both s and t and does not sample any vertex from X. See Figure 4.3 for an illustration.

We have,

$$\Pr(G_i \text{ is good}) \geqslant 1/k^2 \cdot (1-1/k)^{2k-1} \geqslant 1/8k^2. \qquad (\text{as } k \geqslant 2 \text{ so } (1-1/k)^{2k-1} \geqslant (1/2)^3)$$

Given the independence of choices of G_i for $i \in [r]$, we have,

$$\Pr(\text{No } G_i \text{ is good}) \leqslant (1-1/8k^2)^r \leqslant \exp\left(-200k^2 \ln n/8k^2\right) = n^{-25}.$$

Therefore, there is a graph G_{i^*} for $i^* \in [r]$ where s and t are sampled but X is not (see Figure 4.3). This means that the spanning forest T_{i^*} has to contain the edge (s, t) as there is no other path between s and t (we have effectively "deleted" X by not sampling it). Thus, the edge (s, t) belongs to H with probability at least $1 - n^{-25}$. A union bound over all possible pairs $s, t \in G$ concludes the proof.

4.3 The Dynamic Streaming Algorithm

We present our single pass dynamic streaming algorithm for k-vertex-connectivity in this section. The algorithm outputs a certificate of k-vertex-connectivity for the input graph at the end of the stream. Thus, by the definition of the certificate, to know whether or not the input graph is k-vertex-connected, it suffices to test if the certificate is k-vertex-connected, which can be done at the end of the stream using any offline algorithm. The following theorem formalizes Result 4.2.

Theorem 7. *There is a randomized dynamic streaming algorithm that given an integer $k \geqslant 1$ before the stream and a graph $G = (V, E)$ in the stream, outputs a certificate H of k-vertex-connectivity of G with high probability using $O(kn \cdot \log^4 n)$ bits of space.*

This algorithm is just an implementation of Algorithm 4 in dynamic streams. We fix the vertex sets V_i in Algorithm 4 before the stream so the only thing we need to specify is how we compute the spanning forests during the stream. We compute a spanning forest T_i of G_i in the stream using the dynamic streaming algorithm in Proposition 4.1.2 with parameters $N = |V_i|$ and $\delta = n^{-4}$. After the stream, we output the certificate H. This completes the description of the streaming algorithm.

We start by bounding the space of this algorithm.

Lemma 4.3.1. *This algorithm uses $O(kn \cdot \log^4 n)$ bits of space with high probability.*

Proof. During the stream, we run a spanning forest algorithm for each graph G_i for $i \in [r]$. The algorithm of Proposition 4.1.2 with parameters $N = |V_i|$ and $\delta = n^{-4}$ takes at most $c\,|V_i| \cdot \log^3(n^4\,|V_i|)$ bits of space for some absolute constant c. We store $r = O(k^2 \ln n)$ spanning forests so the total space taken is

$$\sum_{i=1}^{r} c\,|V_i| \cdot \log^3(n^4\,|V_i|) \leqslant c \log^3(n^5) \sum_{i=1}^{r} |V_i| = O(kn \log^4 n),$$

where the first inequality uses $|V_i| \leqslant n$ and the second one uses $\sum_{i=1}^{r} |V_i| = O(kn \log n)$ (by Claim 4.2.1). ∎

We are now ready to prove Theorem 7.

Proof of Theorem 7. By Lemma 4.3.1, this algorithm uses $O(kn \cdot \log^4 n)$ bits of space with high probability. The high probability guarantee can be even moved from the space bound to the correctness in the following way: if the space of the algorithms at any point increases beyond this high-probability bound guarantee, we simply terminate the algorithm and output "fail". This happens with negligible probability by Lemma 4.3.1. The algorithm is also correct with high probability by Theorem 6. Moreover, none of the spanning forest algorithms of Proposition 4.1.2 fail with high probability (by a union bound over the failure probabilities of the r spanning forest algorithms). Thus, the streaming algorithm (deterministically) uses $O(kn \cdot \log^4 n)$ bits of space and, by union bound, with high probability outputs a certificate of k-vertex-connectivity. ∎

4.4 The Lower Bound

In this section, we extend the prior single-pass lower bound of [SW15] for vertex connectivity to multi-pass algorithms. The following theorem formalizes Result 4.3.

Theorem 8. *For any integer $p \geqslant 1$, any randomized p-pass insertion-only streaming algorithm that given an integer $1 \leqslant k \leqslant n/2$ before the stream and an n-vertex (multi-)graph $G = (V, E)$ in the stream, outputs whether G is k-vertex connected with probability at least $2/3$, needs $\Omega(kn/p)$ bits of space.*

We use the standard approach of proving lower bounds on the space of streaming algorithms via communication complexity (this is further spelled out in the proof of Theorem 8). The communication lower bound itself is proven using a reduction from the well-known set disjointness problem defined as follows.

Definition 4.4.1 (Set Disjointness (DISJ_N)). *For any integer $N \geqslant 1$, DISJ_N is defined as follows: Alice and Bob are given length N binary strings $x \in \{0,1\}^N$ and $y \in \{0,1\}^N$, respectively. They can communicate back and forth and need to output "No" if there exists an index $i \in [N]$ such that $x_i = y_i = 1$ and "Yes" otherwise. We assume both players have access to a shared source of randomness.*

We use the following standard lower bound on the communication complexity of this problem.

Proposition 4.4.2 ([KS92, Raz90, BYJKS04]). *For any integer $N \geqslant 1$, any two-way randomized protocol for DISJ_N that errs with probability at most $1/3$ needs $\Omega(N)$ bits of communication.*

We use this result to prove a communication complexity lower bound for vertex connectivity.

Proposition 4.4.3. *For any integers $n, k \geqslant 1$ such that $1 \leqslant k \leqslant n/2$ the following is true. Any randomized communication protocol wherein Alice and Bob receive edges of an n-vertex (multi-)graph $G = (V, E)$ partitioned between the two, and can output whether or not G is k-vertex-connected with probability at least $2/3$ requires $\Omega(kn)$ bits of communication.*

Proof. We start with a high level sketch of the proof. We use a reduction from the DISJ_N communication problem for $N = \Theta(kn)$. Alice and Bob construct a bipartite graph G on n fixed vertices and pick their edges based on the values in their input strings x and y in

DISJ_N. G will be constructed in a way that if x and y are disjoint, then G will contain a complete bipartite graph and has vertex connectivity k; otherwise, at least one edge is missing and the graph has vertex connectivity strictly less than k. Thus, solving k-vertex connectivity also solves DISJ_N implying the space lower bound. We now formalize this idea.

To prove the bound for parameters n and k, we start with an instance of DISJ_N such that $N = k \cdot (n-k)$. Alice and Bob construct an n-vertex bipartite graph $G = (L \sqcup R, E)$ with k vertices on L and $n-k$ vertices on R as follows:

- *Vertices:* the vertices in L are $u_1, u_2, \ldots, u_k$ and the vertices in R are $v_1, v_2, \ldots, v_{n-k}$.
- *Edges:* the indices of Alice's string x and Bob's string y can be expressed using coordinates $i \in [k]$ and $j \in [n-k]$ (since $N = k \cdot (n-k)$). If $x_{i,j} = 0$ then Alice has an edge (u_i, v_j) and if $y_{i,j} = 0$ then Bob has an edge (u_i, v_j) (this way, there can be up to two edges between any pairs of vertices).

The following claim is the key part for establishing the correctness of our reduction.

Claim 4.4.4. *G is k-vertex connected iff x and y are disjoint.*

Proof. If x and y are disjoint, then for every $i \in [k], j \in [n-k]$ either $x_{i,j} = 0$ or $y_{i,j} = 0$ and thus edge (u_i, v_j) exists in G. Thus, G contains a complete bipartite graph. Deleting any set X of $k-1$ vertices leave at least one vertex $u_i \in L$ and one vertex $v_j \in R$. Since u_i and v_j are connected, and any vertex in L is connected to v_j and any vertex in R is connected to u_i, we have that $G - X$ is connected. Therefore, G is k-vertex-connected.

If x and y are not disjoint, then there are indices i^* and j^* such that $x_{i^*,j^*} = 1$ and $y_{i^*,j^*} = 1$ implying that edge (u_{i^*}, v_{j^*}) does not exist in G. Deleting all vertices in L except u_{i^*} disconnects v_{j^*} from the rest of the graph. Thus, there is a vertex cut of size $k-1$ implying that G is not k-vertex connected. ∎ Claim 4.4.4

The proof of Proposition 4.4.3 now follows from Proposition 4.4.2: Alice and Bob, given any instance (x, y) of DISJ_N, can construct the graph G in the reduction without any communication and run the protocol for k-vertex-connectivity on G. If the protocol returns G is k-vertex-connected, they return "Yes" and otherwise they return "No". The correctness follows from Claim 4.4.4. This implies that the k-vertex-connectivity protocol needs $\Omega(N) = \Omega(kn)$ communication by Proposition 4.4.2, concluding the proof. ∎

We can now obtain Theorem 8 as a standard corollary of Proposition 4.4.3.

Proof of Theorem 8. Given a p-pass streaming algorithm for the k-vertex-connectivity problem, Alice and Bob can use the algorithm to solve the communication problem as follows. Alice treats her edges in the communication problem as the first part of the stream and Bob treats his edges as the second part of the stream. The players run the streaming algorithm on this stream by communicating the memory content whenever they finish running that pass of the algorithm on their input. This requires sending the memory content for $2p-1$ times until Bob can compute the answer of the streaming algorithm.

Assuming we start with a p-pass streaming algorithm that uses only $o(kn/p)$ bits of space, the above approach gives us a communication protocol with $o(kn)$ communication for k-vertex-connectivity, with the same probability of success as the streaming algorithm. This contradicts Proposition 4.4.3, and concludes the proof of Theorem 8. ∎

We note that this p-pass lower bound can be matched up to polylogarithmic factors under the assumption of a multi-pass streaming model with unbounded intermediate memory (specifically, we require $\widetilde{O}(kn)$ intermediate memory between passes). To achieve this, we run $r/p = O\left(\frac{k^2}{p}\log n\right)$ iterations of Algorithm 4 in each pass. Since each pass only needs to store a $1/p$ fraction of the total space, the space required per pass becomes $\widetilde{O}(kn/p)$.

Chapter 5

Maximum Matching in Dynamic Streams

5.1 Introduction

We study the maximum matching problem in the *dynamic* streaming model. In this problem, the edges of an input graph $G = (V, E)$ are presented to the algorithm as a sequence of both edge insertions and deletions. The goal is to recover an approximate maximum matching of G at the end of the stream using a limited space smaller than the input size, namely, $o(n^2)$ space where n is the number of vertices. The dynamic graph streaming model is highly motivated by applications to processing massive graphs and has been studied extensively in recent years; see, e.g. [AGM12a, AGM12b, KLM+17, Kon15, BHNT15, MTVV15, AKLY16, AKL17, KKP18, NY19, ACK19a, BCG20, KMM+20, DK20, Kon21] and references therein.

A brief note on the history of dynamic streaming matching is in order. Initiated by a breakthrough result of [AGM12a], for most graph problems studied in insertion-only streams, researchers were able to subsequently obtain algorithms with similar guarantees in dynamic streams as well; this includes connectivity [AGM12a], cut sparsifiers [AGM12b], spectral sparsifiers [KLM+17], densest subgraph [MTVV15], subgraph counting [AGM12b], $(\Delta+1)$-vertex coloring [ACK19a], among many others. This placed the maximum matching problem in a rather unique position in the literature: while there is a straightforward 2-approximation algorithm for this problem in insertion-only streams using only $O(n \log n)$ space [FKM+04], no non-trivial approximation algorithms were developed for this problem even in $o(n^2)$ space, despite significant attention; see, e.g. [Ber14, CCHM15, CCE+15].

This problem was addressed in a series of (independent and concurrent) work [Kon15, AKLY16, CCE+16]. In particular, [AKLY16] proved that any α-approximation algorithm for matching in dynamic streams requires $\Omega(n^{2-o(1)}/\alpha^3)$ bits of space and designed an α-approximation algorithm with $O(n^2/\alpha^3 \cdot \text{polylog}\,(n))$ space for this problem ([Kon15] gave a slightly weaker lower and upper bounds for this problem and [CCE+16] obtained an algorithm with similar performance as [AKLY16]). The work of [AKLY16] thus brought the gap between space upper and lower bounds on this problem down to an $n^{o(1)}$ factor. The lower bound of [AKLY16] relied on a remarkable characterization of dynamic streaming algorithms due to [LNW14, AHLW16] that allows for transforming linear sketching lower bounds to dynamic streams. However, this characterization requires making strong requirements from the streaming algorithms (such as processing doubly exponentially long streams); see [KP20] for a detailed discussion on this topic. More recently, [DK20] bypassed this characterization step entirely and along the way, improved the lower bound for this problem to $\Omega(n^2/\alpha^3)$ space directly in dynamic streams. This constitutes the state-of-the-art for the dynamic streaming matching problem.

In parallel to this line of work on the matching problem that focused on determining the "high order terms" in the space complexity of this problem (namely, up to $n^{o(1)}$ or $\text{polylog}\,(n)$ factors), there has also been substantial work on determining the "lower order terms" on space complexity of other dynamic graph streaming problems [SW15, KNP+17, NY19, Yu21]. For instance, [NY19], building on [KNP+17], proved that any dynamic streaming algorithm for connectivity requires $\Omega(n \cdot \log^3(n))$ space which matches the algorithm of [AGM12a] up to constant factors. This quest for obtaining asymptotically optimal bounds is common in the streaming literature beyond graph streams such as in frequency moment estimation [KNW10a, KNW10b, LW13, ANPW13, BKSV14], empirical entropy [HNO08, CCM10, JW13], numerical linear algebra [CW09], compressed sensing [PW11, PW13], and sampling [KNP+17].

This state-of-affairs is the motivation behind our work: *Can we determine the space complexity of the maximum matching problem down to its lower order terms?* We resolve this question in the affirmative by presenting an improved algorithm for this problem.

Main Result (Formalized in Theorem 9)**.** *There is a dynamic streaming algorithm that with high probability outputs an α-approximation to maximum matching using $O(n^2/\alpha^3)$ space for any $\alpha \ll n^{1/2}$.*

Let us right away note that the condition of $\alpha \ll n^{1/2}$ in our main result is not arbitrary[1]:

[1]The same condition is used in all prior lower bounds in [Kon15, AKLY16, DK20] as well as algorithms [Kon15, CCE+16] with the exception of algorithm of [AKLY16].

for $\alpha > n^{1/2}$, we have $n^2/\alpha^3 < n/\alpha$, while one needs $\Omega((n/\alpha) \cdot \log n)$ space simply to store an α-approximate matching! As a result, our algorithm now matches the lower bound of [DK20] up to constant factors in almost the entirety of its meaningful regime for parameter α, thus completely resolving the space complexity of the maximum matching problem in dynamic streams. We now discuss further aspects of our work.

Beyond L_0-samplers. The key technique in dynamic graph streams is the use of L_0-samplers[2] that allow for sampling an edge from an stream that contains both insertions and deletions of the edges (see Section 2.3).

Previously-best algorithms of [AKLY16,CCE+16] for dynamic streaming matching sample $O(n^2/\alpha^3)$ edges from the input graph (from a carefully-designed non-uniform distribution) and show that this sample contains an α-approximate matching. For the sampling, they need to use L_0-samplers that will bring in an additional polylog (n) factor overhead in the space. At the same time, a careful examination of the lower bound of [DK20] suggests that one needs to recover $\Omega(n^2/\alpha^3)$ *edges* from the graph (not only *bits*, assuming one only communicates edges). On top of this, the lower bound of [NY19] for the connectivity problem is based on showing that recovering $(n-1)$ edges of a spanning forest in the input, essentially require paying the cost of $(n-1)$ L_0-samplers as well, leading to their $\Omega(n \cdot \log^3(n))$ lower bound. Putting all this together, it is natural to conjecture that one also needs $\Omega(n^2/\alpha^3 \cdot \text{polylog}(n))$ space for the matching problem[3].

Our algorithm in this paper is still based on finding $\Theta(n^2/\alpha^3)$ edges from the input graph. It turns out however that one can do this more efficiently than using the same number of L_0-samplers. In particular, we show a way of recovering these edges with only $O(1)$ bit overhead per edge *on average*. This is achieved using a novel sketching primitive in this paper (Section 5.2.2). On a high level, this sketch allows us to recover *sparse induced subgraphs* of the input graph, specified to the algorithm only at the end stream, in a more efficient manner than recovering them one edge at a time via L_0-samplers[4]. We believe this idea can be useful for obtaining asymptotically optimal algorithms for other dynamic streaming problems as well.

Classifying input graphs. Another key idea in our paper is a way of roughly classifying input graphs into "easy" and "hard" instances. Informally speaking, the easy instances

[2] We are only aware of a single work [KLM+17] in dynamic graph streams that does not use L_0-samplers.

[3] This was in fact the authors' conjecture at the beginning of this project.

[4] Let us note that our sketch cannot do magic: The problem of finding sparse induced subgraphs is at the core of the lower bound approaches for dynamic streaming matching in [AKLY16, DK20], thus there is no hope of solving it "efficiently". Our sketch shows that one can recover these graphs without paying any extra cost *over* the lower bounds of these work.

are the ones that one can recover a large matching from them by sampling $\ll n^2/\alpha^3$ edges (again, in a non-uniform way). Such a graph can then be handled in $O(n^2/\alpha^3)$ space even if we use L_0-samplers for our sampling given we now need much fewer number of samples than before. One of our two main lemmas (Lemma 5.3.2) gives one characterization of these graphs: essentially, any "hard" graph, i.e., a one not solvable by the above approach, includes a subgraph on $n - o(n/\alpha)$ vertices with only $\approx n$ edges and a matching of size $\approx n - o(n/\alpha)$ (they essentially have an induced matching of size $\approx n - o(n/\alpha)$). A reader familiar with [Kon15, AKLY16, DK20] may notice that this family precisely captures the graphs in prior lower bounds for dynamic streaming matching problem.

Our next main lemma (Lemma 5.3.3) then gives an algorithm for solving these hard graphs. The idea behind the algorithm is as follows. Let S denote the vertices in the induced sparse subgraph of the input and let T be the remaining vertices (we will be able to recover an approximate version of this partitioning *at the end* of the stream). *If* we are able to recover edges inside S, we will be done as there is a large matching in S and it does not have too many extra edges. The problem is that we will not know this set until the end of the stream and by that point we should have collected all the required information. This is where our main sketching tool mentioned earlier comes into place. Informally, the sketch allows us to, for any vertex $v \in S$, recover the neighbors $N(v)$ of v using roughly $|N(v) \cap T| + (|N(v) - T| \cdot \text{poly}\log(n))$ bits (as opposed to $|N(v)| \cdot \text{polylog}(n)$ bits via L_0-samplers). As the total number of edges outside T is quite small, i.e., $\approx n$ in total, this is a huge saving for us that allows for obtaining our desired $O(n^2/\alpha^3)$ bit upper bounds.

We shall remark that in this discussion, we have been imprecise to give a rough intuition of our approach; the actual details turn out to be considerably more challenging as described in Section 5.4 and Section 5.5.

"Shaving" log-factors? Finally, our improvement over prior work in [AKLY16, CCE+16] at no place is obtained via "shaving log-factors". Indeed there is a considerable gap of $n^{\Omega(1)}$ factor between the parameters that our easy-graph algorithms and hard-graph algorithms can still handle within $O(n^2/\alpha^3)$ space. This in turn allowed us to be quite cavalier with the parameters (e.g., using $\log n$-factors or $n^{\Omega(1)}$-factors where constant or polylog(n) sufficed) and still recover an optimal space bound.

Standard Sketching Toolkit. A standard tool we use is sparse recovery to solve the following problem.

Problem 2 (Sparse Recovery)**.** *Given an integer $k \geqslant 1$ and a vector $x \in \mathbb{R}^m$ specified in a dynamic stream with the promise that $\|x\|_0 \leqslant k$, recover all of x at the end of the stream.*

We use the following result on sparse recovery over finite fields.

Proposition 5.1.1 (c.f. [DV13]). *Let q be any prime number and $k \geqslant 1$ be an arbitrary integer. There is a deterministic (poly-time computable) linear sketch, called* `Sparse-Recovery`*, for Problem 2 for k-sparse vectors $x \in \mathbb{F}_q^m$, with size*

$$s_{\texttt{SR}} = s_{\texttt{SR}}(m, k, q) = O(k \cdot \log m \cdot \log q)$$

bits that always outputs the correct answer on k-sparse vectors. Moreover, all computations of this linear sketch are also performed over the field $\mathbb{F}_q$.

We shall note that for our application, we actually need the 'moreover' part of Proposition 5.1.1 (which limits the use of more standard sparse recovery approaches).

5.2 New Sketching Toolkit

We present two novel linear sketches in this section that are needed for our main algorithm. The first one is a simple way of sampling random edges from a group of vertices to obtain an edge to a random neighbor of this set (as opposed to a random edge). The second (and main[5]) linear sketch is a sparse-recovery-type sketch that allows for finding neighborhood of a vertex (or group of vertices) assuming we already know a set that intersects largely with the neighborhood.

5.2.1 Neighborhood-Edge Sampler

Suppose we have a group S of vertices, and we want to sample a vertex v from the neighborhood of S. If we want the probability of sampling v to be proportional to $\deg(v)$, we can sample an edge incident on S (using an `L0-Sampler`) and return the other endpoint; but what if we would like to sample v uniformly at random from $N(S)$? There is a simple (and standard) solution for this problem using an `L0-Sampler` *if we do not need to recover the edge incident on v*[6]. However, for our purpose, we crucially need the edge as well therefore just an `L0-Sampler` will not work. We formulate the following problem to address this formally.

Problem 3. *Given a graph $G = (V, E)$ specified in a dynamic stream, and a set $S \subseteq V$ of vertices at the start of the stream, output an edge (u, v) such that $u \in S$ and v is sampled uniformly at random from $N(S)$.*

[5]The reason we consider this the most important of our sketches is that essentially *all* our saving of $\operatorname{poly}\log(n)$ factors comes from the efficiency of this sketch. For the first sketch, even a somewhat loose (in terms of extra $\operatorname{poly}\log(n)$ factors) bound in the space suffices for our purpose.

[6]Create an n-dimensional vector where entry i denotes the number of edges incident on v_i from S; then use an `L0-Sampler` to return an element from the support of this vector uniformly at random.

We design a linear sketch for solving this problem.

Lemma 5.2.1. *There is a linear sketch, called* `NE-Sampler`(G, S)*, for Problem 3 with size*

$$s_{\texttt{NES}} = s_{\texttt{NES}}(n) = O(\log^3 n)$$

bits, that outputs FAIL *with probability at most* $1/100$ *and gives a wrong answer with probability at most* n^{-8}*.*

To solve Problem 3, we first need the following standard lemma. The proof of this lemma is known and is presented only for completeness.

Lemma 5.2.2. *There is a linear sketch called* `NE-Counter` *of size* $O((\log n) \cdot \log(1/\delta_E))$ *bits that given a graph* $G = (V, E)$ *presented in a dynamic stream, and any two sets* S *and* T *of vertices at the beginning of the stream, outputs whether or not* $|N(S) \cap T| = 1$ *with probability of error at most* δ_E*.*

Proof. The algorithm is simply as follows. For $i = 1$ to $t = \log(1/\delta_E)$ iterations:

(*i*) Pick a pair-wise independent hash function $h_i : [n] \to \{1, 2\}$.
For $j \in \{1, 2\}$, let $T_{ij} := \{v \in T \mid h_i(v) = j\}$.

(*ii*) Count the number of edges from S to T_{i1} and from S to T_{i2} using counters c_{i1} and c_{i2}. If both counters are zero or both non-zero, then return $|N(S) \cap T| \neq 1$.

If the algorithm never terminated up until here, output $|N(S) \cap T| = 1$.

The algorithm uses $O(\log n \cdot \log(1/\delta_E))$ space as by Proposition 2.4.7, it only needs $O(\log n)$ bits per iteration to store each hash function (and another $O(\log n)$ bits for the counters). Moreover, whenever the algorithm returns $|N(S) \cap T| \neq 1$, the answer is correct: Either S has zero neighbors in $T_{i1} \cup T_{i2} = T$, or it has non-zero number of edges from S to both T_{i1} and T_{i2} implying S has more than one neighbor in T. The only case in which the algorithm can make an error is when $|N(S) \cap T| > 1$, but it does not detect it.

Consider any pair of vertices $u \neq v$ in $N(S) \cap T$. In each iteration $i \in [t]$, the probability that u and v hash to the same value is half since h_i is a pairwise-independent hash function. Thus, the probability that this event happens in all $t = \log(1/\delta_E)$ iterations is δ_E, which means the algorithm can only err with probability at most δ_E. ∎

We are now ready to prove Lemma 5.2.1 using the following linear sketch.

Algorithm 5. `NE-Sampler`(G, S): A linear sketch for Problem 3.

Input: A graph $G = (V, E)$; a set $S \subseteq V$ of vertices.

Output: An edge from S to a uniformly random vertex of $N(S)$.

Sketching matrix:

1. Repeat the following for $k = 50$ iterations:
2. Do in parallel $i \in \{0, 1, \ldots, \lceil 2 \log n \rceil\}$:
   (a) Let $h_i : V \to [2^i]$ be a pairwise independent hash function. Define $T_i := \{v \in V \mid h_i(v) = 1\}$ so that each vertex belongs to T_i with probability $1/2^i$ (with pairwise independence across vertices).
   (b) Store an `L0-Sampler` $\mathcal{L}_i$ with parameters $\delta_E = n^{-10}$ and $\delta_F = 1/100$ for edges between S and T_i. Also, store a `NE-Counter` (Lemma 5.2.2) for (S, T_i) with $\delta_E = n^{-10}$.

Recovery:

1. Go over all copies of `NE-Counter` in an arbitrary order and find one that returns "1" as the size of the intersection. Extract an edge from the corresponding `L0-Sampler` and output it.
2. If no `NE-Counter` returns "1" then output "FAIL".

We now show the correctness of `NE-Sampler`. Let $d := |N(S)|$ and let i^* be such that $2^{i^*} \leqslant 4d < 2^{i^*+1}$. Consider the iteration i^* wherein we sample each vertex in T_{i^*} with probability $1/2^{i^*}$. Let X be a random variable denoting the number of elements of $N(S)$ that are sampled. Let X_j be the random variable which is 1 if the j-th vertex of $N(S)$ is sampled and 0 otherwise. We have $X = \sum_{j \in N(S)} X_j$. We want to find the probability that $X = 1$:

$$\begin{aligned}
\Pr(X = 1) &= \sum_{j \in N(S)} \Pr(X_j = 1 \wedge X_{-j} = 0) = \sum_{j \in N(S)} \Pr(X_j = 1) \cdot \Pr(X_{-j} = 0 \mid X_j = 1) \\
&= \sum_{j \in N(S)} \frac{1}{2^{i^*}} \cdot (1 - \Pr(X_{-j} \geqslant 1 \mid X_j = 1)) \qquad \text{(by the choice of } T_{i^*}\text{)}
\end{aligned}$$

$$\geqslant \sum_{j \in N(S)} \frac{1}{2^{i^*}} \cdot (1 - \mathbb{E}\left[X_{-j} \mid X_j = 1\right]) \qquad \text{(by Markov inequality)}$$

$$= \sum_{j \in N(S)} \frac{1}{2^{i^*}} \cdot \left(1 - \sum_{j' \in N(S) \setminus j} \mathbb{E}\left[X_{j'}\right]\right)$$

(by linearity of expectation and pairwise independence of $X_j, X_{j'}$ for $j \neq j' \in N(S)$)

$$= \frac{d}{2^{i^*}} \cdot \left(1 - \frac{d-1}{2^{i^*}}\right) \qquad \text{(as } |N(S)| = d\text{)}$$

$$\geqslant \frac{1}{4} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{8}. \qquad \text{(as } 2^{i^*} \leqslant 4d < 2^{i^*+1}\text{)}$$

For the purpose of analysis we say that the algorithm fails if the parallel iteration i^* fails. This could happen if $X \neq 1$ or if the corresponding `L0-Sampler` fails. By the above calculation and the bound on $\delta_F = 1/100$, we get that the failure probability is at most $7/8 + 1/100 < 9/10$. Thus, the probability that all $k = 50$ iterations fail is at most $(0.9)^{50} < 0.01$. Therefore, exactly one element v from $N(S)$ is picked in some iteration, and we can find an edge from S to v with probability at least 0.99. Also, we can union bound over the error probabilities of $O(k \cdot \log n)$ copies of `L0-Sampler` and `NE-Counter` giving a total error probability of at most n^{-8}. This proves the correctness of `NE-Sampler` as required in Lemma 5.2.1.

The space taken by an `L0-Sampler` is $O(\log^2 n)$ bits since $\delta_E = n^{-10}$ and $\delta_F = 1/100$ (using Proposition 2.3.1). The space taken by one `NE-Counter` (Lemma 5.2.2) is $O(\log^2 n)$ bits, and by a hash function is $O(\log n)$ bits. As we run $O(\log n)$ copies of `L0-Sampler`, `NE-Counter` and hash functions in parallel, the total space taken by `NE-Sampler` is $O(\log^3 n)$ bits implying Lemma 5.2.1.

5.2.2 Sparse-Neighborhood Recovery

The second problem we would like to tackle is a sparse recovery type problem: suppose we have a group S of vertices, and at the end of the stream, we (somehow) managed to find a *superset* T of all but a "tiny" fraction of vertices in $N(S)$. Can we recover the remainder of $N(S) - T$ efficiently using our sketch? Formally,

Problem 4 (Sparse-Neighborhood Recovery)**.** *Let $a, b \geqslant 1$ be known integers such that $a \geqslant 100b$. Consider a graph $G = (V, E)$ specified in a dynamic stream and let $S \subseteq V$ be a known subset of vertices. The goal is to, given a set $T \subseteq V$ at the end of the stream, return the set $N(S) - T$, assuming the following promises:*

(*i*) *size of* T *is at most* a*;*

(*ii*) *size of* $N(S) - T$ *is at most* b*;*

(*iii*) *for every vertex* $v \in N(S) - T$*, we have* $|S \cap N(v)| < c$*.*

In words, in Problem 4, we have a set S of vertices, known at the *start* of the stream, and we are interested in their neighbors *outside* a given set T, specified at the *end* of the stream. Our guarantees are roughly that T is not "too large" (parameter a), neighborhood of S outside T is "small" (parameter b), and each vertex outside T only has "few" neighbors inside S (parameter c). See Figure 5.1 for an illustration.

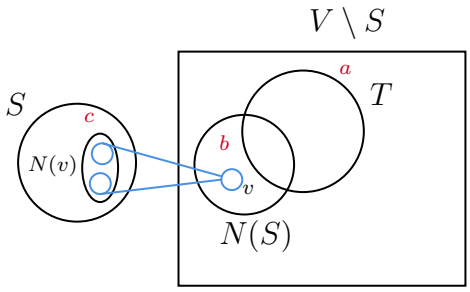


Figure 5.1: This figure shows S and its neighborhood $N(S)$ which intersects with T. T has size at most a and $N(S) - T$ has size at most b. Also, every vertex v in $N(S) - T$ has at most c neighbors in S.

Lemma 5.2.3. *There is a linear sketch, called* `SN-Recovery`(G, S)*, for Problem 4 that uses sketch and randomness of size, respectively,*

$$s_{\texttt{SNR}} = s_{\texttt{SNR}}(n, a, b, c) = O(a \cdot \log c + b \cdot \log n \cdot \log c) \quad \text{and} \quad O(a \cdot \log n)$$

bits and outputs a wrong answer with probability at most $1 - 4\exp(-\dfrac{b}{24 \log n})$ *for* $b \geqslant 24 \log n$*.*

The key part of Lemma 5.2.3 is that the dependence on $\log n$ is only on the (much) smaller b-term, as opposed to the a-term (otherwise, this result would be immediate by Proposition 5.1.1[7]). This saving is a key factor in the success of our algorithms in achieving asymptotically optimal bounds for the matching problem.

[7]What makes Problem 4 particularly different from sparse-recovery is that since we only know T is a superset of $N(S)$ and not equal to it, our underlying vector is only $(a + b)$-sparse as opposed to b-sparse.

We present two algorithms for solving Lemma 5.2.3. The first one is very simple and already achieves the asymptotic optimal bounds on the sketch size (in terms of parameters a, b); the problem with this approach however is that the recovery algorithm for the sketch requires an exhaustive search of all options and thus requires exponential time in the worst case; the sketching matrix of the algorithm also requires $O(a \cdot n)$ bits of space to store which is prohibitively large for our purpose. Thus, we present this sketch as a warm-up in Section B.1. Our second sketch is more involved but uses a near-linear time recovery algorithm and not too much randomness[8]. We present this algorithm in the remainder of this section.

To continue, we give a different representation of Problem 4 that makes the exposition simpler.

Vector-representation of Problem 4. For any graph $G = (V, E)$ and set $S \subseteq S$ of vertices, define the n-dimensional vector $x = x(G, S)$, indexed by vertices in V, such that for all $i \in [n]$,

$$x_i := \begin{cases} 0 & \text{if } v_i \in S \\ \sum_{u \in S} \mathbb{I}[(u, v_i) \in E] & \text{otherwise} \end{cases}. \tag{5.1}$$

A basic observation is that $\text{supp}(x(G, S))$ (set of non-zero entries of $x(G, S)$) corresponds to $N(S)$ in G. The second observation is that each update to an edge (u, v) in a dynamic stream $\mathbf{vec}(E)$ can be directly used to update x as well. Finally, throughout the proof, we will take q to be the smallest prime larger than c and work with the field $\mathbb{F}_q$, i.e., the field of integers modulo q. Given Promise *(iii)* of Problem 4, x will have the same non-zero entries among the coordinates in S still even in $\mathbb{F}_q$. As such, our goal is to design a linear sketch Φ such that one can recover $\text{supp}(x)$ from $\Phi \cdot x$ (interpreted in $\mathbb{F}_q$) with high probability.

Proof of Lemma 5.2.3. The high-level overview of the proof is as follows. We will first design a sketch of size $O(\log q)$ bits only that can recover the value of x_i for some random $i \in T$. This sketch however may fail with constant probability and err with probability roughly b/a introduced by elements in the support of x outside T. We will then use $O(a)$ of these sketches in parallel with each other to recover a constant fraction of x projected on T using a sketch of size $O(a \log q)$. As a result, this effectively shrinks the size of the set T for us that we need to focus next. However, due to the potential error introduced by the sketches, this means that we may now need to recover $O(b)$ *additional* elements from

[8]The randomness used by this sketch is still larger than the sketch size which is problematic on the surface for us. However, we will be able to reuse this randomness across multiple sketches and thus achieve our desired bounds on the space overall.

outside of this new T. A bit more formally, this approach allows us to find a vector y and shrink the set T to another set T_y such that $|T_y| \leqslant |T|/4$ and $x - y$ only has $b + O(b)$ elements outside T_y. This means that we need to solve the original problem, on the vector $x - y$ now, with a smaller parameter a but a larger parameter b.

Our approach is thus to run this recursive algorithm *non-adaptively* by storing appropriate sketches of x only and since we know y, use linearity of sketches to compute the sketch of $x - y$ also. The key part of the proof is to ensure that we shrink size of T, i.e., parameter a, *rapidly* while grow b *slowly* in this process, all while keeping the total sketch size still only $O(a \log q)$. The slow growth of b thus allows us reach a situation where the resulting vector we have to work with becomes $\Theta(b)$-sparse overall (inside and outside of the current set T). We can then use standard sparse recovery on this vector to recover it entirely using another sketch of size $O(b \log q \cdot \log n)$ which will give us the desired bound on the sketch size.

We now start the formal proof. The first step is a subroutine for recovering a single index in T with a small error probability. In the rest of the proof, we assume that size of T is exactly a without loss of generality (say, by increasing n slightly and adding dummy elements to T).

Lemma 5.2.4. *There is a linear sketch,* `Index-Recovery`(x, T, a, b, γ)*, that given a vector $x \in [n]$ and set $T \subseteq [n]$ and parameters $a \geqslant 100b$ (as specified in Problem 4), plus a confidence parameter $\gamma \in (0, 1/4)$, returns an index $i \in T$ chosen uniformly at random together with the value of x_i. The probability of failure of the algorithm, δ_F, and the probability it outputs a wrong value for x_i, δ_E, are*

$$\delta_F \leqslant \frac{4}{5} \qquad \text{and} \qquad \delta_E \leqslant \gamma \cdot \frac{b}{2a} + \frac{b^2}{2a^2}.$$

The sketch has size $O(\log(1/\gamma) \cdot \log q)$ bits and requires $O(\log(1/\gamma) \cdot \log n)$ random bits.

Proof. Let $h : [n] \to [2a]$ be a pair-wise independent hash function and define $H := \{i \in [n] \mid h(i) = 1\}$ (so that each index belongs to H with probability $1/2a$ and the choice of vertices is pairwise independent). Compute $z := \sum_{i \in H} x_i$. At the end of the stream, if $|H \cap T| \neq 1$, terminate and output FAIL. Moreover, for $j = 1$ to $t := \log(1/\gamma)$ iterations:

(*i*) Pick a pairwise independent hash function $h_j : H \to \{1, 2\}$.
For $k \in \{1, 2\}$, let $H_{jk} := \{i \in H \mid h_j(i) = k\}$ (so that H_{j1} and H_{j2} form a partition of H with each index in H having sent to each one uniformly).

(*ii*) For $k \in \{1, 2\}$, compute $z_{jk} := \sum_{i \in H_{jk}} x_i$. If both z_{jk} are non-zero terminate and output FAIL.

If the algorithm never terminated, output i where $\{i\} := H \cap T$ and $x_i = z$ as the answer. This concludes the description of the `Index-Recovery` algorithm.

We now analyze correctness and bound size and randomness of `Index-Recovery`. Firstly,

$$\Pr(|H \cap T| = 1) = \Pr\big(\text{there exists a } \underline{\text{unique}} \text{ index } i \in T \text{ with } h(i) = 1\big) \geqslant \frac{1}{2} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{4},$$

using the pairwise independence of h, exactly as in the proof of Lemma 5.2.1. Note that the algorithm can detect this event exactly as it knows both H and T at the end of the stream. Moreover, by union bound,

$$\begin{aligned}\Pr(|H \cap (N(S) \setminus T)| = 0) &= 1 - \Pr(\text{there is an index } i \in (N(S) \setminus T) \text{ with } h(i) = 1) \\ &\geqslant 1 - \frac{b}{2a} \geqslant 1 - \frac{1}{200}.\end{aligned}$$

Conditioned on both events above, for $i \in T$ where $\{i\} = H \cap T$, we get that $z = x_i$. Moreover, for every iteration $j \in [t]$ in the algorithm, it can never be that both z_{j1} and z_{j2} are non-zero. Thus, conditioned on these events, the algorithm outputs a correct answer. This in particular means that the probability the algorithm outputs FAIL is at most $3/4 + 1/200 < 4/5$ as required.

Now note that the only way the algorithm may output a wrong answer is when $|H \cap T| = 1$ but $|H \cap (N(S) \setminus T)| > 0$. Firstly, using that h is a pairwise independent hash function, we have that,

$$\Pr(|H \cap (N(S) \setminus T)| \geqslant 2) \leqslant \binom{|N(S) \setminus T|}{2} \cdot \frac{1}{4a^2} \leqslant \left(\frac{e \cdot b}{2}\right)^2 \cdot \frac{1}{4a^2} < \frac{b^2}{2a^2}. \qquad (\text{as } |N(S) \setminus T| \leqslant b)$$

Thus, whenever this event happens, we can simply charge the error to the $b^2/2a^2$ term in δ_E in the lemma statement. In the following, we only need to handle the case when $|H \cap (N(S) \setminus T)| = 1$.

For the error to happen in this case, we should have that the single element in $H \cap T$ and the single element in $H \cap N(S) \setminus T$ are always mapped the same by the hash function h_j for every iteration $j \in [t]$. The probability of this event happening is exactly $1/2^t = \gamma$ by

the pairwise independence of each h_j (and their independence across t iterations). Thus, conditioned on the event that $H \cap N(S) \setminus T$ has size 1, which itself happens with probability at most $b/2a$ calculated above, the probability of error is γ. This is also accounted for in the $\gamma \cdot b/2a$ term in δ_E, concluding the proof of the correctness.

As for the space, storing each of the hash functions requires $O(\log n)$ and each sum $O(\log q)$ bits. Given that we have $O(\log(1/\gamma))$ iterations, we get the desired space bound. ∎

We now build on the sketch `Index-Recovery` in Lemma 5.2.4 to recover a constant fraction of the indices of $x \in \mathbb{R}^n$ in T again with small error.

Lemma 5.2.5. *There is a linear sketch,* `Partial-Recovery`(x, T, a, b, γ)*, that given vector $x \in \mathbb{R}^n$ and set $T \subseteq [n]$ and parameters $a \geqslant 100b$ (as specified in Problem 4), plus a confidence parameter $\gamma \in (0, 1/4)$, returns a vector $y \in \mathbb{R}^n$ and a set $T_y \subseteq [n]$ with the following properties with probability at least $1 - 2\exp(-\gamma b)$:*

(i) T_y is a subset of T and has size at most $a/4$.

(ii) $x - y$ has at most $b + 12 \cdot (\gamma b + \frac{b^2}{a})$ non-zero elements outside T_y.

The sketch has size $O(a \cdot \log(1/\gamma) \cdot \log q)$ bits and requires $O(a \cdot \log(1/\gamma) \cdot \log n)$ random bits.

Proof. Initialize $T_y \leftarrow T$ and $y \leftarrow \mathbf{0}^n$. For $t := (5a \cdot \ln 8) < 12a$ iterations: Run `Index-Recovery`(x, T, a, b, γ) independently and, if the output is *not* FAIL, let $y_i = x_i$ for $i \in T$ and x_i returned by the sketch, and update $T_y \leftarrow T_y \setminus \{i\}$. This concludes the description of `Partial-Recovery`.

Fix any index $i \in T$. In any iteration that `Index-Recovery` does not output FAIL, the probability that i is returned as the index is $1/a$ by Lemma 5.2.4. Thus, the probability that this index i is never returned through all the iterations is at most

$$\left(1 - \left(\frac{1}{5} \cdot \frac{1}{a}\right)\right)^t \leqslant \exp\left(-\frac{1}{5} \cdot \frac{t}{a}\right) = \exp(-\ln 8) = \frac{1}{8}. \quad \text{(by the choice of } t = (5a \cdot \ln 8))$$

For any $i \in T$, define an indicator random variable X_i which is 1 iff i still belongs to T_y at the end of `Partial-Recovery`. Let $X = \sum_{i \in T} X_i$ denote the size of T_y. By the above calculation, we have $\mathbb{E}[X] \leqslant a/8$. Moreover, consider the t *independent* random variables

denoting the randomness of each iteration of the algorithm. We have that X is a 1-Lipschitz function of these variables (as changing randomness of one iteration, can only make a single index i join or leave T_y). Thus, by McDiarmid's inequality (Proposition 2.4.5),

$$\Pr(|T_y| > a/4) \leqslant \Pr(|X - \mathbb{E}[X]| > a/8) \leqslant 2 \cdot \exp\left(-\frac{2\,a^2}{64t}\right) < \exp\left(-\frac{a}{400}\right). \qquad \text{(by the choice of } t < 12a\text{)}$$

Now again consider an index $i \in T$. For $x_i \neq y_i$ to happen, we should have that the iteration in which `Index-Recovery` returns i makes an error. Thus, the number of indices of $x - y$ that are different (outside T_y), is upper bounded by the number of iterations wherein `Index-Recovery` errs. Each iteration makes an error with probability at most $\gamma \cdot \frac{b}{2a} + \frac{b^2}{2a^2}$ by Lemma 5.2.4. Thus, letting Y denote the random variable for the number of erroneous iterations, we get that

$$\mathbb{E}[Y] \leqslant t \cdot \left(\gamma \cdot \frac{b}{2a} + \frac{b^2}{2a^2}\right) \leqslant 6 \cdot (\gamma \cdot b + \frac{b^2}{a}). \qquad \text{(by the choice of } t < 12a\text{)}$$

Moreover, since Y is a sum of t independent random variables (one per iteration), by Chernoff bound,

$$\Pr\left(Y \geqslant 12 \cdot (\gamma \cdot b + \frac{b^2}{a})\right) \leqslant \Pr(Y \geqslant 2\,\mathbb{E}[Y]) \leqslant \exp(-\gamma \cdot b).$$

As there are at most b indices outside T that are non-zero in x (but zero in y), we get that $x - y$ has at most $b + 12 \cdot (\gamma \cdot b + \frac{b^2}{a})$ non-zero entries outside T_y with probability at least $1 - \exp(-\gamma \cdot b)$.

Combining the above two bounds, and since $a \geqslant 100b$ and $\gamma < 1/4$, we get the final probability bound on the properties of the algorithm. The space and randomness are also $t = O(a)$ times that of `Index-Recovery`, which implies the lemma by the bounds in Lemma 5.2.4. ∎

We now present the sketching matrix of `SN-Recovery` and postpone its recovery algorithm to later.

Algorithm 6. The **sketching matrix** of `SN-Recovery`(G, S) and its corresponding sketches.

1. Define the recursive sequences $\{\gamma_j\}$, $\{a_j\}$, and $\{b_j\}$ for any $j \geqslant 1$:

$$\gamma_1 = \frac{1}{24}, \quad \gamma_j := \frac{\gamma_{j-1}}{2} \quad \text{and} \quad a_1 := a, \quad a_j := \frac{a_{j-1}}{4}.$$

$$b_1 = b, \quad b_j := b_{j-1} + 12 \cdot (\gamma_{j-1} \cdot b_{j-1} + \frac{b_{j-1}^2}{a_{j-1}}).$$

2. For $j = 1$ to t iterations where t is the minimum of $\log \log n$ and the last index where $a_t \geqslant 100 \cdot b_t$:

   (a) Let Φ_j be the sketching matrix of `Partial-Recovery`$(\cdot, \cdot, a_j, b_j, \gamma_j)$ (the sketching matrix is independent of first two arguments of the input and thus we do not provide those arguments).

3. Let Φ^* be the sketching matrix of `Sparse-Recovery` to recover a $(a_t + b_t)$-sparse vector in $\mathbb{F}_q^n$.

4. Compute the sketches $\{\Phi_j \cdot x\}_{j=1}^t$ and $\Phi^* \cdot x$ for the vector representation x of $N(S)$ defined earlier.

We bound the sketch size and randomness of `SN-Recovery` in the following claim.

Claim 5.2.6. *The sketch and randomness of* `SN-Recovery` *in Algorithm 6 have size*

$$O((a + b \cdot \log n) \cdot \log q) \quad \textit{and} \quad O(a \cdot \log n)$$

bits, respectively.

Proof. By Lemma 5.2.5, we have that $\Phi_j \cdot x$ requires $O(a_j \cdot \log(1/\gamma_j) \cdot \log q)$ size and $O(a_j \cdot \log(1/\gamma_j) \cdot \log n)$ bits of randomness, respectively. Thus, the total size of the sketches $\{\Phi_j \cdot x\}_{j=1}^t$ is

$$O(\log q) \cdot \sum_{j=1}^{t} a_j \cdot \log(1/\gamma_j) = O(\log q) \cdot \sum_{j=1}^{t} \frac{a_1}{4^j} \cdot \log(2^j/\gamma_1) = O(a_1 \cdot \log(1/\gamma_1) \cdot \log q),$$

where the second to last equality is by explicitly computing the recursive definition of a_j, γ_j, and the last one is since the series $\sum_{j=1}^{\infty} j/4^j$ converges to some constant. Considering that

$a_1 = a$ and γ_1 is a constant, we get that the total size of sketches $\{\Phi_j \cdot x\}_{j=1}^{t}$ is $O(a \cdot \log q)$. The same exact calculation, by plugging in the randomness bound of Lemma 5.2.5 instead, also gives an $O(a \cdot \log n)$ bound on the number of random bits.

We now need to also calculate the size of Φ^* (this part of the sketch does not involve any randomness). By Proposition 5.1.1, size of $\Phi^* \cdot x$ is

$$O((a_t + b_t) \cdot \log n \cdot \log q). \tag{5.2}$$

We start by bounding the value of b_t. By the recursive definition of γ_j, a_j, b_j, we have that for all $j \in [t]$

$$\begin{aligned} b_j &= b_{j-1} + 12 \cdot (\gamma_{j-1} \cdot b_{j-1} + \frac{b_{j-1}^2}{a_{j-1}}) = b_{j-1} \cdot \left(1 + 12 \cdot \frac{\gamma_1}{2^{j-1}} + 12 \cdot \frac{b_{j-1}}{a_1}\right) \\ &\leqslant b_{j-1} \cdot \left(1 + 24 \cdot \frac{\gamma_1}{2^j} + 12 \cdot \frac{b_t}{a_{j-1}}\right). && \text{(as } b_j\text{'s are increasing)} \end{aligned}$$

As such, for b_t itself, we have that,

$$\begin{aligned} b_t &\leqslant b_1 \cdot \prod_{j=1}^{t} \left(1 + 24 \cdot \frac{\gamma_1}{2^j} + 12 \cdot \frac{b_t}{a_{j-1}}\right) && \text{(by the inequality above)} \\ &\leqslant b_1 \cdot \exp\left(\sum_{j=1}^{t} 24 \cdot \frac{\gamma_1}{2^j} + 12 \cdot \frac{b_t}{a_{j-1}}\right) && \text{(as } (1+x) \leqslant \exp(x) \text{ for all } x > 0) \\ &\leqslant b_1 \cdot \exp(24 \cdot \gamma_1) \cdot \exp\left(12 b_t \cdot \sum_{j=1}^{t} \frac{1}{a_{j-1}}\right) && \text{(as } \textstyle\sum_{j=1}^{\infty} 1/2^j = 1) \\ &= b_1 \cdot \exp(24 \cdot \gamma_1) \cdot \exp\left(12 \cdot \frac{b_t}{a_t} \cdot \sum_{j=1}^{t} \frac{1}{4^{t-j+1}}\right) && \text{(as } a_{j-1} = a_1/4^{j-1} = 4^{t-j+1} \cdot a_t) \\ &\leqslant b_1 \cdot \exp(24 \cdot \gamma_1) \cdot \exp\left(4 \cdot \frac{b_t}{a_t}\right) && \text{(as } \textstyle\sum_{j=1}^{t} 1/4^{t-j+1} \leqslant 1/3) \\ &\leqslant b_1 \cdot e^2. && \text{(as } \gamma_1 = 1/24 \text{ and } a_t \geqslant 100 b_t) \end{aligned}$$

Thus, even though b_j's are increasing, we still have $b_t = O(b_1)$.

As for the value of a_t, we have that $a_t = a_1/4^t$. Thus, either $t = \log\log n$ and so we get $a_t \leqslant a_1/\log n$, or $a_t < 400 \cdot b_{t+1}$, which implies $a_t = O(b_t) = O(b_1)$ in this case. This implies that $a_t = O(a_1/\log n + b_1)$.

Plugging in the bounds on a_t, b_t in Eq (5.2), we get that the size for $\Phi^* \cdot x$ is $O(a \cdot \log q + b \cdot \log n \cdot \log q)$. This concludes the proof. ∎

We now show how to use these sketches to perform the recovery part.

Algorithm 7. The **recovery algorithm** of `SN-Recovery`(G, S).

1. Let $x_1 = x$ and $T_1 = T$. For $j = 1$ to t iterations (where t is the parameter in Algorithm 6):

   (a) Use $\Phi_j \cdot x$ to obtain the vector y_j and set $T_{j+1} := T_{y_j}$ of `Partial-Recovery`$(x_j, T_j, a_j, b_j, \gamma_j)$ and let $x_{j+1} = x_j - y_j$ (here, and throughout the proof, a_j, b_j, γ_j are as defined in Algorithm 6).

   This step can be done by exploiting the linearity of the sketches to compute

   $$\Phi_j \cdot x_j = \Phi_j \cdot (x_{j-1} - y_{j-1}) = \Phi_j \cdot x - \sum_{k=1}^{j-1} \Phi_j \cdot y_k,$$

   as the algorithm has already calculated $\Phi_j \cdot x$, and knows $y_1, \ldots, y_{j-1}$ explicitly.

2. Use $\Phi^* \cdot x$ to compute $\Phi^* \cdot x_t$ as specified above and recover x_t by `Sparse-Recovery`. As we have all $y_1, \ldots, y_t$ also, we can compute $x = x_t + \sum_{j=1}^{t} y_j$ which recovers $N(S)$ accordingly.

The following lemma proves the correctness of the algorithm.

Claim 5.2.7. *With probability* $1 - 4\exp\left(-\frac{b}{24 \log n}\right)$*, for every* $j \in [t]$*,* T_j *has size at most* a_j *and* x_j *has at most* b_j *non-zero entries outside* T_j*.*

Proof. We prove this by induction wherein we assume that the high probability event of Lemma 5.2.5 happens every time we invoke it – we then bound the probability that this does not happen explicitly.

For $j = 1$, the claim statement holds trivially by the promise of Problem 4. Now suppose this is the case for some index j. At this point, by Lemma 5.2.5, when invoking `Partial-Recovery`$(x_j, T_j, a_j, b_j, \gamma_j)$, we get that the resulting pair of vector y_j and set T_{y_j} have the following properties:

(i) T_{y_j} is a subset of T_j with size at most $a_j/4$. Hence, $T_{j+1} = T_{y_j}$ has size at most $a_{j+1} = a_j/4$ also.

(*ii*) $x_j - y_j$ has at most $b_j + 12(\gamma_j b_j + \frac{b_j^2}{a_j})$ non-zero elements outside T_{y_j}. Hence, $x_{j+1} = x_j - y_j$ has at most $b_{j+1} = b_j + 12(\gamma_j b_j + \frac{b_j^2}{a_j})$ non-zero elements outside of $T_{j+1} = T_{y_j}$ also.

This proves the induction step.

We now need to also account for the error probability of each application of Lemma 5.2.5, which, for an iteration $j \in [t]$, is at most $2\exp(-\gamma_j \cdot b_j)$. As such, by union bound, probability of error is at most

$$\begin{aligned} 2 \cdot \sum_{j=1}^{t} \exp(-\gamma_j \cdot b_j) &\leqslant 2 \cdot \sum_{j=1}^{t} \exp\left(-\frac{\gamma_1}{2^j} \cdot b_1\right) && \text{(as } \gamma_j = \gamma_1/2^j \text{ and } b_j \geqslant b_1\text{)} \\ &\leqslant 4 \cdot \exp\left(-\frac{\gamma_1}{2^t} \cdot b_1\right) \\ &\text{(as the largest term in the series dominates the sum of the rest)} \\ &\leqslant 4 \cdot \left(-\frac{b}{24 \cdot \log n}\right), \end{aligned}$$

by the choice of γ_1 and since $t \leqslant \log\log n$. ∎

Conditioned on the event of Claim 5.2.7, we have that at the end of the last iteration, x_t is $(a_t + b_t)$-sparse. Thus, by the guarantee of `Sparse-Recovery` in Proposition 5.1.1, the algorithm correctly recovers x_t. As $x = x_t + \sum_{j=1}^{t} y_j$ and the algorithm has already computed y_j's also, it will recover x correctly. This, combined with the bounds on the sketch size and randomness in Claim 5.2.6, concludes the proof of Lemma 5.2.3.

Remark 5.2.8. The number of random bits needed by the `SN-Recovery` sketch is $O(a \log n)$ which is more than our budget to store individually for each sketch. But we can reuse these random bits for all copies of `SN-Recovery` sketches and union bound over the failure probability. Thus, we use and store at most $O(a \log n)$ random bits over *all* `SN-Recovery` sketches that we use in our dynamic streaming algorithm for matching.

5.2.3 Neighborhood-Size Tester

Before we move on from this section, we also mention a simple helper sketch that allows to approximately verify if the promises of Problem 4 are satisfied for a given input. We formally define the problem as follows:

Problem 5 (Neighborhood-Size Testing)**.** *Let $a, \tilde{b} \geqslant 1$ be known integers such that $a \geqslant 16\tilde{b}$. Consider a graph $G = (V, E)$ specified in a dynamic stream and let $S \subseteq V$ be a known subset of vertices. The goal is to, given a set $T \subseteq V$ at the end of the stream, return "Yes" if $|N(S) - T| \leqslant \tilde{b}$ and "No" if $|N(S) - T| \geqslant 2\tilde{b}$, assuming the following promises:*

(i) size of T is at most a;

(ii) $|N(S) - T| \leqslant \tilde{b}$ or $|N(S) - T| \geqslant 2\,\tilde{b}$;

In words, in Problem 5, we have a set S of vertices, known at the *start* of the stream, and we are interested in the size of the neighborhood *outside* a given set T, specified at the *end* of the stream. We guarantee that T is not "too large" (parameter a) and that a is slightly larger than $\tilde{b}$ and want to know the size of the neighborhood of S outside T. If the size is between $\tilde{b}$ and $2\tilde{b}$ then the answer can be arbitrary.

Lemma 5.2.9. *There is a linear sketch, called* `NE-Tester`(G, S)*, for Problem 5 that uses sketch and randomness of size*

$$s_{\texttt{NET}} = s_{\texttt{NET}}(n, a, \tilde{b}) = O\left(\frac{a}{\tilde{b}} \cdot \log^3 n\right)$$

bits and with high probability outputs the correct answer.

The solution to this problem is standard and is included for completeness. We will solve this problem by sampling random neighbors (using `L0-Sampler`) and see how many of them lie outside T. Note that all vertices in T may not be neighbor to S, but we can fix that by adding artificial edges from all vertices of T to S only for the tester. We can then count the number of neighbors picked by the neighborhood samplers outside T and get an estimate of the number of neighbors outside T. Formally,

Algorithm 8. `NE-Tester`(G, S): A linear sketch for Problem 5.

Input: A graph $G = (V, E)$ specified via $\mathbf{vec}(E)$, a set $S \subseteq V$ specified at the beginning of the stream, and a set $T \subseteq V$ specified at the end of the stream.

Output: "Yes" if $|N(S) - T| \leqslant \tilde{b}$ and "No" if $|N(S) - T| \geqslant 2\tilde{b}$.

Parameters: Let $k := \frac{100}{99} \cdot 150 \ln n \cdot \left(\frac{a+\tilde{b}}{\tilde{b}}\right)$.

Sketching matrix:

1. Given T at the end of the stream, add edges from all vertices in T to an arbitrary vertex in S to the stream for this tester.
2. Sample k copies of `L0-Sampler` for $N(S)$ with parameters $\delta_E = n^{-10}$ and $\delta_F = 1/100$.

Recovery:

1. Go over the k copies of `L0-Sampler` that do not fail and extract a vertex from them.
2. If the number of sampled vertices outside T is at most $200 \log n$, output "Yes"; otherwise "No".

We now analyze the algorithm. Let $Y := N(S) \setminus T$ and $y := |Y|$. We consider the cases when $y \leqslant \tilde{b}$ and $y \geqslant 2\tilde{b}$ separately in the following.

Claim 5.2.10. *If $y \leqslant \tilde{b}$ then the algorithm outputs "Yes" correctly with probability at least $1 - n^{-4}$.*

Proof. A sample lies outside T with probability $\frac{y}{a+y} \leqslant \frac{\tilde{b}}{a+\tilde{b}}$ and does not fail with probability 0.99 by the guarantee of `L0-Sampler`. For any sample $i \in [k]$, let Z_i be an indicator random variable which is 1 iff the sampled vertex is outside T. We thus have

$$\mathbb{E}[Z_i] \leqslant 0.99 \cdot \frac{\tilde{b}}{a+\tilde{b}} = \frac{150 \log n}{k}. \qquad \text{(by the choice of } k \text{ in the algorithm)}$$

Let $Z := \sum_{i=1}^{k} Z_i$ denote the number of sampled vertices outside of T. As Z_i's are independent of each other, by Chernoff bound (Proposition 2.4.3) with $\mu_H = 150 \log n \geqslant \mathbb{E}[Z]$ and $\varepsilon = 1/3$, we have,

$$\Pr(Z > 200 \log n) = \Pr(Z > (1+\varepsilon) \cdot \mu_H) \leqslant \exp\left(-\frac{150 \ln n}{36}\right) < n^{-4},$$

which concludes the proof, as when the number of sampled vertices that lie outside T is at most $200 \log n$, the algorithm outputs "Yes". ∎

We now consider the complementary case when $y \geqslant 2\tilde{b}$.

Claim 5.2.11. *If $y \geqslant 2\tilde{b}$ then the algorithm outputs "No" with probability at least $1 - n^{-3}$.*

Proof. A sample lies outside T with probability $\frac{y}{a+y} \geqslant \frac{2\tilde{b}}{a+2\tilde{b}}$ and does not fail with probability 0.99 by the guarantee of `L0-Sampler`. For any sample $i \in [k]$, let Z_i be an indicator random variable which is 1 iff the sampled vertex is outside T. We thus have

$$\mathbb{E}[Z_i] \geqslant 0.99 \cdot \frac{2\tilde{b}}{a+2\tilde{b}} \geqslant 0.99 \cdot \frac{2\tilde{b}}{a+\tilde{b}} \cdot \frac{7}{8} \geqslant \frac{250 \log n}{k}.$$

(by the choice of k in the algorithm and since $a \geqslant 16\tilde{b}$)

Let $Z := \sum_{i=1}^{k} Z_i$ denote the number of sampled vertices outside of T. As Z_i's are independent of each other, by Chernoff bound (Proposition 2.4.3) with $\mu_L = 250 \log n \geqslant \mathbb{E}[Z]$ and $\varepsilon = 1/5$, we have,

$$\Pr(Z \leqslant 200 \log n) = \Pr(Z \leqslant (1-\varepsilon) \cdot \mu_H) \leqslant \exp\left(-\frac{150 \ln n}{50}\right) < n^{-3},$$

which concludes the proof, as when the number of sampled vertices that lie outside T is more than $200 \log n$, the algorithm outputs "No". ∎

Therefore, we showed that we can distinguish between the two cases. To find the error probability we can union bound over the error probabilities of both cases and the error probabilities of all copies of `L0-Sampler` and conclude that the error probability is at most $2n^{-3}$ (since there are at most n^2 copies of `L0-Sampler`).

The algorithm uses k copies of `L0-Sampler` each of which has size $O(\log^2 n)$ bits (Proposition 2.3.1 with $\delta_E = n^{-10}$ and $\delta_F = 1/100$). Thus, the sketch size is $k \cdot O(\log^2 n) = O\left(\frac{a}{\tilde{b}} \log^3 n\right)$ bits, proving Lemma 5.2.9.

5.3 Main Result and Setup

In this section, we present our main results for α-approximating the maximum matching of any given graph in dynamic streams using $O(n^2/\alpha^3)$ bits of space. Specifically, we prove the following theorem for linear sketches which immediately gives a dynamic streaming algorithm with the same guarantees by Proposition 2.2.4.

Theorem 9. *There is a linear sketch that given any parameter $\alpha \leqslant n^{1/2-\delta}$ for any constant $\delta > 0$, and any n-vertex graph $G = (V, E)$ specified via $\mathbf{vec}(E)$, with high probability outputs an α-approximate maximum matching of G using $O(n^2/\alpha^3)$ bits of space.*

We will make the following (more or less standard) assumptions when designing our algorithms. Both assumptions are made for simplicity of exposition and we show how to remove them later in this section.

Assumption 1 (Knowledge of matching size)**.** *At the beginning of the stream, we are given an estimate* opt *with the promise that the maximum matching size of the input graph G has size at least* opt*. The goal is then to return a matching of size $(\eta_0 \cdot \text{opt}/\alpha)$ for some absolute constant $\eta_0 > 0$ at the end of the stream.*

Assumption 2 (Range of parameters)**.** *We assume that the parameter* opt *of Assumption 1 and approximation factor α satisfy the following equations:*

$$\text{opt} \geqslant \alpha^2 \cdot n^{\delta} \qquad \text{and} \qquad \alpha > 100.$$

Remark 5.3.1. While we assume Assumption 1 and Assumption 2 when designing our algorithms, even if these assumptions are *not* satisfied, the algorithms (with high probability) will not output an edge that does not belong to the graph, but may output a matching that is not sufficiently large for our purpose.

The plan for designing our main algorithms is then to focus on the problem of Assumption 1 (and further assume Assumption 2). We first give a linear sketch that can handle "easy" graphs for this problem. In particular, we prove the following lemma.

Lemma 5.3.2 (Match-Or-Sparsify Lemma)**.** There is a linear sketch that given any graph $G = (V, E)$ specified via $\mathbf{vec}(E)$, uses $O(\text{opt}^2/\alpha^3)$ bits of space and with high probability outputs a matching M_{easy} that satisfies at least one of the following conditions:

- **Match-case:** The matching M_{easy} has at least $(\text{opt}/8\alpha)$ edges;
- **Sparsify-case:** The induced subgraph of G on vertices not matched by M_{easy}, denoted by $\overline{G}_{\text{easy}}$, has at most $(20\,\text{opt} \cdot \log^4 n)$ edges and a matching of size at least $3\,\text{opt}/4$.

This lemma should be interpreted as follows: we can either find a matching of size $(\text{opt}/8\alpha)$ (thus already solve the problem of Assumption 1 with $\eta_0 = 1/8$), or certify that we had a "hard" graph to work on. Our main saving in the space then comes from the subsequent algorithm that handles any input that leads to the sparsify-case of Lemma 5.3.2. We prove Lemma 5.3.2 in Section 5.5.

We note that Lemma 5.3.2 bears similarities to the so-called "residual sparsity property" of greedy matching established in [ACG+15] (see also [Kon18b]). In this context, those results prove that if one samples $\approx \mathsf{opt}^2/\alpha^3$ edges of the graph uniformly at random, and compute a maximal matching of the sample greedily, then the induced subgraph of G on unmatched vertices have maximum degree $\approx \alpha^3 \log n$ with high probability, thus $O(\alpha^3 \cdot n \log n)$ edges in total. Our Lemma 5.3.2 uses a non-uniform sampling method and exploits the fact that the resulting matching is small (otherwise we are in the match-case), to bound the total number of edges in the induced subgraph of unmatched vertices more strongly by $\approx \mathsf{opt} \cdot \mathrm{polylog}\,(n)$. Finally, the non-uniform sampling method used in this lemma is inspired by prior work on dynamic streaming matching algorithms in [AKLY16, CCE+16] although the analysis of the algorithm is quite different.

The following lemma is the heart of the proof. We emphasize that the information provided by algorithm of Lemma 5.3.2 will only be available to the algorithm of this lemma *at the end of the stream* as we have to run both algorithms in parallel in a single pass.

Lemma 5.3.3 (Algorithm for Sparsify-Case)**.** There is a linear sketch that given any graph $G = (V, E)$ specified via $\mathbf{vec}(E)$, uses $O(\mathsf{opt}^2/\alpha^3)$ bits of space and with high probability, given the matching M_{easy} of Lemma 5.3.2 in the recovery step, can recover a matching of size $(\mathsf{opt}/8\alpha)$ in G.

Lemma 5.3.3 gives an efficient way of solving "hard instances" of the dynamic streaming matching problem, namely, the ones left out by our more standard approach in Lemma 5.3.2. This lemma is where we use our `SN-Recovery` sketches in place of L_0-samplers and is the source of efficiency of our general algorithm. We prove Lemma 5.3.3 in Section 5.4.

Theorem 9 now follows easily from Lemma 5.3.2 and Lemma 5.3.3 by lifting Assumption 1 and Assumption 2.

Proof of Theorem 9. By Lemma 5.3.2 and Lemma 5.3.3, we can use $O(\mathsf{opt}^2/\alpha^3)$ bits of space under Assumption 1 and Assumption 2 and find a matching of size at least $(\mathsf{opt}/8\alpha)$ in G with high probability.

Removing Assumption 2. Firstly, if $\mathsf{opt} < \alpha^2 \cdot n^\delta$, then by the promise of Theorem 9 that $\alpha < n^{1/2-\delta}$, we get that $\mathsf{opt} < n^{1-\delta}$. At this point, even if we run an algorithm with $O((\mathsf{opt}^2/\alpha^3) \cdot \mathrm{poly}\log(n))$ space, it will still be $o(n^2/\alpha^3)$ bits as required by Theorem 9. Thus, we can run any of the previously-best algorithms for this problem, e.g. the ones in [AKLY16, CCE+16], to solve the problem.

Secondly, if $\alpha \leqslant 100$, we can simply maintain a counter mod two between every pairs of vertices to store all edges of G in $O(n^2)$ bits of space which is permitted by Theorem 9 when $\alpha = O(1)$. This allows us to solve the problem exactly.

Removing Assumption 1. Let $\mathcal{A}(\text{opt}, \alpha)$ be the algorithm we obtained so far under Assumption 1. We simply run $\mathcal{A}(o, \beta)$ for all choices of $o \in \{2^i \mid i = 0 \text{ to } \log n\}$ and $\beta = (\alpha/2\eta_0)$ in parallel and return the largest matching found. By Remark 5.3.1, these matchings all belong to the input graph with high probability and for the choice of $o \geqslant \mu(G)/2$, where $\mu(G)$ is the maximum matching size of G, we can apply our results for $\text{opt} = o$ to get a matching of size $\eta_0 \cdot \text{opt}/(\beta/2) = \mu(G)/\alpha$ in the graph, which is precisely an α-approximation as desired. Finally, the space of this new algorithm is

$$O(1) \cdot \sum_{o \in \left\{2^i \mid i \in [\log n]\right\}} \frac{o^2}{\alpha^3} = O(n^2/\alpha^3) \text{ bits}$$

as the sum is forming a geometric series. This concludes the proof of Theorem 9. ∎

We conclude this section by making the following remark about the sketches we use.

Remark 5.3.4. Throughout our main algorithms in the remainder of the paper, we use at most $O(n^2)$ copies of the sketching primitives `NE-Tester`, `NE-Sampler` and `SN-Recovery` developed in Section 5.2. For all these sketches the probabilities of failure and error are $1/100$ and n^{-10}, respectively. We can simply do a union bound over all these sketches and have that with a high probability, none of them are going to err. Hence, we condition on this high-probability event here and do not explicitly account for the error probability of this part each time. However, we will consider the case that (some of) these sketches may output FAIL still.

5.4 Main Algorithm: Handling the Sparsify-Case

As the main part of our work in this paper is the algorithm in Lemma 5.3.3, we change the order of presentation and start with this algorithm and postpone the proof of Lemma 5.3.2 to the next section.

Lemma 5.3.2 allows us to find a matching M_{easy} which is either large enough, or the subgraph induced by its unmatched vertices is sparse and has a large matching. Our task now is to handle the latter case efficiently, i.e., prove Lemma 5.3.3. We emphasize that we can only know this particular sparse subgraph of the input *after* the pass over the input, and by that point we should have collected all the required information from the graph already. The following lemma is a slightly weaker version of Lemma 5.3.3.

Lemma 5.4.1 (Slightly weaker version of Lemma 5.3.3)**.** *There is a linear sketch that given any graph $G = (V, E)$ specified via* $\mathbf{vec}(E)$*, with high probability uses $O(\text{opt}^2/\alpha^3)$ bits of space and given the matching M_{easy} of Lemma 5.3.2 in the recovery step that satisfies the sparsify-case, can recover a matching of size $(\text{opt}/8\alpha)$ in G with probability at least $(1 - n^{-\delta/6})$ and does not output any edge that is not in G with high probability.*

Let us show that Lemma 5.4.1 immediately proves Lemma 5.3.3 in its full generality. Firstly, it is without loss of generality to assume that M_{easy} satisfies the sparsify-case as otherwise, the algorithm can simply return M_{easy} itself which is of size at least $(\text{opt}/8\alpha)$ in the match-case and satisfies the promise of Lemma 5.3.3.

Secondly, to improve the success probability to a high-probability bound, we can run the algorithm of above lemma in parallel for $(60/\delta) = O(1)$ times and return the largest matching output by any copy. With high probability, the algorithm still does not output an edge not in the graph and uses $O(\text{opt}^2/\alpha^3)$ bits of space (as $\delta = \Theta(1)$). The probability that none of these matchings are large enough is only $\left(n^{-\delta/6}\right)^{60/\delta} = n^{-10}$; thus, the algorithm also outputs a large enough matching with high probability. This proves Lemma 5.3.3 assuming Lemma 5.4.1. As such, in this section, we focus on proving Lemma 5.4.1.

To simplify the exposition, we present and analyze the sketching matrix and recovery step of the linear sketch in Lemma 5.4.1 separately.

5.4.1 The Sketching Matrix

The sketching matrix of Lemma 5.4.1 is computed as follows. We create $k \approx \text{opt}/\alpha$ groups of vertices and each group is obtained by sampling each vertex independently with probability $1/k$ (so vertices can belong to more than one group or none at all). We connect these groups using a fixed (k/α)-regular graph and throughout the stream, we only focus on the edges appearing between vertices of connected groups. Over these edges then, we maintain one `NE-Tester` and one `SN-Recovery` for each group with parameters $a \approx (\text{opt}/\alpha^2)$, $b \approx \tilde{b} \approx n^{\delta}$, and $c = \Theta(1)$. Moreover, to save space in the sketching matrices, we use the same set of random bits for sketching matrices of all `SN-Recovery` copies. This amounts to a total of $O(\text{opt}^2/\alpha^3)$ bits of space.

Algorithm 9. The sketching matrix of Lemma 5.4.1.

Input: A graph $G = (V, E)$ specified via $\mathbf{vec}(E)$.

Parameters: Let $k := 10\text{opt}/\alpha$, $a = 2\text{opt}/\alpha^2$, $\tilde{b} = n^{\delta/4}$, and $c = 30/\delta$.

(*i*) Create a collection of **groups** of vertices $\mathcal{V} := (V_1, \ldots, V_k)$ as follows: For each $i \in [k]$, independently sample a $(\log^2 n)$-wise independent hash function $h_i : V \to [k]$ and set $V_i := \{v \in V \mid h_i(v) = 1\}$.

(*ii*) Let $F \in \{0,1\}^{k \times k}$ be the adjacency matrix of an arbitrarily fixed (k/α)-regular graph on $[k]$ (with no self-loops or parallel edges). We say that two groups V_i and V_j are **neighbor** whenever $F(i,j) = 1$.

(*iii*) For any group $i \in [k]$, define the subgraph $G(V_i)$ on vertices V but only consisting of edges between V_i and its neighbor-groups, i.e., with edges $\{(u,v) \in E \mid u \in V_i,\ v \in V_j,\ F(i,j) = 1,\ \forall j\}$.

(*iv*) For every $i \in [k]$, return sketching matrices of `SN-Recovery`$(G(V_i), V_i)$ with parameters $a, b = 2\tilde{b}, c$ and `NE-Tester`$(G(V_i), V_i)$ with parameters a and $\tilde{b}$ as the final sketching matrix – to save space, use the *same random bits* for sketching matrices of all copies of `SN-Recovery`.

We note that in Algorithm 9, each vertex of V may appear in more than one group of $\mathcal{V}$ or no group at all. We start by bounding the size of the sketching matrix and the extra information stored by Algorithm 9.

Lemma 5.4.2. *Algorithm 9 uses $O(\text{opt}^2/\alpha^3)$ bits of space with high probability.*

Proof. Line (*i*) requires storing $k = O(\text{opt}/\alpha)$ different $(\log^2 n)$-wise independent hash functions, each of which requiring $O(\log^3 n)$ bits by Proposition 2.4.7. This is bounded by $O(\text{opt}^2/\alpha^3)$ bits by Assumption 2.

Line (*ii*) does *not* require storing F explicitly as it is fixed and input-independent (we can use any standard way of generating a fixed (k/α)-regular graph[9]).

Line (*iii*) and the graphs it works with are deterministic functions of $\mathbf{vec}(E)$ in the stream and the groups stored in Line (*i*). Note that we are *not* going to store these subgraphs in the stream but rather for each update (u,v) to $\mathbf{vec}(E)$, we only update all subgraphs $G(V_i)$ for $i \in [k]$ by checking whether (u,v) also belongs to $G(V_i)$ for $i \in [k]$. Thus, we require no further storage in this line.

[9]For instance, to generate a $2d$-regular graph on N vertices connect vertex i to vertices in $[i-1, i-d]$ and in $[i+1, i+d]$. To generate a $2d+1$-regular graph connect i to $i + N/2$ in addition to the previous vertices. All the calculations are done mod N. Note that N has to be even in the $2d+1$ case which is okay for us because we have $10 \cdot (\text{opt}/\alpha)$ groups.

Line (iv) stores sketching matrix of k copies of `SN-Recovery` with the same parameters $a, b = 2\tilde{b}, c$ and k copies of `NE-Tester` with the same parameters $a, \tilde{b}$. By Lemma 5.2.3, each `SN-Recovery` sketch will take $O((a + \tilde{b}\log n)\log c)$ bits and by Lemma 5.2.9, each `NE-Tester` takes $O(\frac{a}{\tilde{b}} \cdot \log^3 n)$ bits. Both of these are $O(\text{opt}/\alpha^2)$ bits by Assumption 2. As we are storing $k = O(\text{opt}/\alpha)$ of these sketches, the total space will then be $O(\text{opt}^2/\alpha^3)$ as desired. Finally, since we share the randomness of copies of `SN-Recovery`, we only need $O(a \log n)$ bits *in total* which is a lower-order term. ∎ Lemma 5.4.2

By the sparsify-case, we get a sparse graph $\overline{G}_{\text{easy}}$ with a large matching. We identify edges of this matching with certain properties that make them easy to recover while accounting for a constant fraction of the matching. In our subsequent recovery algorithm we will show that we recover a superset of these edges. We now analyze Algorithm 9 and describe the useful properties of certain edges. To continue, we need some notation and definitions.

Notation. We say an edge $e = (u, v)$ **appears between** two groups $V_i, V_j \in \mathcal{V}$ iff $u \in V_i$ and $v \in V_j$, and V_i and V_j are neighbor groups, i.e., $F(i, j) = 1$. Similarly, we say e **appears inside** a group $V_i \in \mathcal{V}$ if there exists some group V_j such that e appears between (V_i, V_j). We write '$e \in (V_i, V_j)$' or '$e \in V_i$' when e appears between (V_i, V_j) or inside V_i, respectively.

Definition 5.4.3 (Group definitions)**.** *For each group $V_i \in \mathcal{V}$, we say that V_i is:*

- **clean** *if it does not contain any vertex of M_{easy}.*
- **expanding** *if more than $\tilde{b}$ edges of $\overline{G}_{\text{easy}}$ appear inside V_i and* **non-expanding** *otherwise.*

Let M be the matching of size at least $3\,\text{opt}/4$ in $\overline{G}_{\text{easy}}$ as guaranteed by Lemma 5.3.2 (recall that $\overline{G}_{\text{easy}}$ is the subgraph of G induced on vertices not matched by M_{easy}). We define $M^\star$ as the following subset of M on "low-degree" vertices of $\overline{G}_{\text{easy}}$, namely:

$$M^\star := \left\{(u, v) \in M \mid \text{each of } u \text{ and } v \text{ has at most } (\tilde{b}/2) \text{ neighbors in } \overline{G}_{\text{easy}}\right\}. \tag{5.3}$$

We will focus on recovering edges of $M^\star$ (which we show are sufficiently many). For this, we need to define several conditions for each edge $(u, v) \in M^\star$ that if satisfied, allows us to recover this edge via our recovery algorithm using the sketches stored by Algorithm 9.

Definition 5.4.4 ($M^\star$-edges definitions)**.** *For any edge e of $M^\star$, we say that e is:*

– **weakly-represented** *by pairs of groups* $V_i \neq V_j \in \mathcal{V}$ *iff:*

  (*i*) *e appears between* V_i *and* V_j *(this means* V_i *and* V_j *has to be neighbor groups),*

  (*ii*) *no edge of* $\overline{G}_{\text{easy}}$ *other than e appears between* V_i *and* V_j*, and*

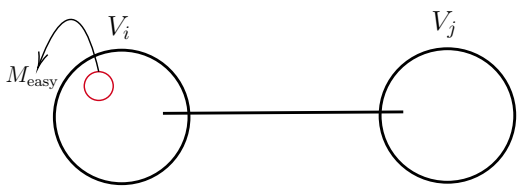
  (*iii*) *both* V_i *and* V_j *are clean.*

– **strongly-represented** *by pairs of groups* $V_i \neq V_j \in \mathcal{V}$ *iff:*

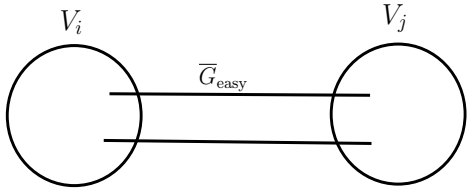
  (*i*) *e is weakly-represented by* (V_i, V_j)*, and*

  (*ii*) *both of* V_i *and* V_j *are non-expanding.*

Figures 5.2 and 5.3 give illustrations of this definition.



(a) This figure shows groups V_i and V_j with exactly one edge e of $\overline{G}_{\text{easy}}$ between them but e is not weakly-represented because V_i contains a vertex of M_{easy}



(b) This figure shows two groups V_i and V_j with multiple edges of $\overline{G}_{\text{easy}}$, none of which can be weakly-represented.

Figure 5.2: Illustration of edges that satisfy the first condition, but not other conditions of weakly-represented edges.

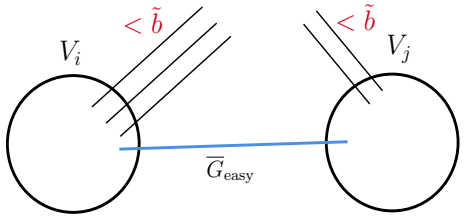


Figure 5.3: This figure shows a strongly-represented edge (in blue). There is exactly one edge e of $\overline{G}_{\text{easy}}$ between V_i and V_j both of which are non-expanding and clean. Thus, e is strongly-represented.

In this subsection, we show that $\approx \text{opt}/\alpha$ edges of $M^\star$ are strongly represented. Then, in the next subsection, we design our recovery algorithm in a way that can recover all

strongly represented edges with high probability. Since these edges are coming from a matching themselves, this allows us to find a large enough matching in the input graph. We now state the main lemma for this subsection.

Lemma 5.4.5. *The number of strongly-represented edges is at least* $\text{opt}/8\alpha$ *with probability at least* $1 - n^{-\delta/6}$.

We start the proof with an easy claim that lower bounds the size of $M^\star$.

Claim 5.4.6. *There are at least* $2\,\text{opt}/3$ *edges in* $M^\star$.

Proof. Recall that M is the $(3\,\text{opt}/4)$-size matching of $\overline{G}_{\text{easy}}$ and $M^\star$ is a subset of M on vertices with degree at most $(\tilde{b}/2)$ in $\overline{G}_{\text{easy}}$. For any $v \in V(M)$, let $d(v)$ denote the degree of v in $\overline{G}_{\text{easy}}$. We have

$$\sum_{v \in V(M)} d(v) \leqslant 2 \cdot \left|E(\overline{G}_{\text{easy}})\right| = 40 \cdot \text{opt} \cdot \log^4 n.$$

(by the sparsify-case of $\overline{G}_{\text{easy}}$ in Lemma 5.3.2)

Thus, the average degree of vertices in $V(M)$ is at most $(40 \log^4 n) = o(\tilde{b})$. By Markov bound, the total number of vertices in $V(M)$ with degree more than $(\tilde{b}/2)$ is then at most $o(\text{opt})$. Removing all these vertices still leaves out $3\,\text{opt}/4 - o(\text{opt}) > 2\,\text{opt}/3$ edges which all belong to $M^\star$. ∎ Claim 5.4.6

Our goal is now to show that $\approx 1/\alpha$ fraction of edges of $M^\star$ are strongly represented by some pairs of groups. In the following, we first bound the probability that an edge is weakly-represented and prove that the number of weakly-represented edges is both large enough and concentrated. We will then bound the number of these edges that will be strongly-represented also (which no longer is necessarily concentrated).

Lemma 5.4.7. *For any edge* $e = (u, v) \in M^\star$,

$$\Pr(e \text{ is weakly-represented by some pairs of groups}) \geqslant \frac{1}{3\alpha}.$$

To prove Lemma 5.4.7, we bound the probability of each condition of being weakly-represented separately.

The first condition is that e should appear between some (V_i, V_j). Given that there are $\approx k^2/\alpha$ pairs neighboring groups (by choice of F) and e can appear between each of these groups with probability $1/k^2$, we will get that the probability e appear between a pair of groups is $\approx 1/\alpha$. Formally,

Claim 5.4.8 (Condition (i) of weakly-represented)**.**

$$\Pr(e \text{ appears between some pairs of groups}) \geqslant \frac{2}{5\alpha}.$$

Proof. Since F is an adjacency matrix of a (k/α)-regular graph on k vertices, there are exactly $(k^2/2\alpha)$ pairs of neighboring groups in $\mathcal{V}$ (assuming both directions of pairs are included). Note that e could appear between multiple groups, so we need to avoid over-counting. In the following, the summands are always only over neighboring pairs. By inclusion-exclusion principle, we have,

$$\begin{aligned}\text{LHS of Claim 5.4.8} &\geqslant \sum_{(V_i,V_j)} \Pr(e \in (V_i,V_j)) - \sum_{(V_{i_1},V_{j_1})\neq(V_{i_2},V_{j_2})} \Pr(e \in (V_{i_1},V_{j_1}) \wedge e \in (V_{i_2},V_{j_2})) \\ &= \sum_{(V_i,V_j)} \frac{1}{k^2} - \sum_{\substack{(V_{i_1},V_{j_1})\neq(V_{i_2},V_{j_2}) \\ |\{i_1,j_1\}\cap\{i_2,j_2\}|=1}} \frac{1}{k^3} - \sum_{\substack{(V_{i_1},V_{j_1})\neq(V_{i_2},V_{j_2}) \\ |\{i_1,j_1\}\cap\{i_2,j_2\}|=0}} \frac{1}{k^4},\end{aligned}$$

where each term of the second inequality is because: for the first-term, the probability depends on the choice of $h_i(u)$ and $h_j(v)$ which are independent; for the second-term, the probability depends on the choice of $h_{i_1}(u)$ and $h_{j_1}(v)$ and $h_{j_2}(v)$ (assuming $i_1 = i_2$, and similarly for other cases) which are independent; and for the last-term, the probability depends on $h_{i_1}(u), h_{j_1}(v)$ and $h_{i_2}(u), h_{j_2}(v)$ which are all independent; as these hash functions are marginally uniform over $[k]$, we get the bound.

Moreover, there are exactly $(k^2/2\alpha)$ choices for the first summand, at most $3k^3/\alpha^2$ for the second one, and at most k^4/α^2 for the last one. Thus,

$$\text{LHS of Claim 5.4.8} \geqslant \frac{k^2}{2\alpha}\cdot\frac{1}{k^2} - \frac{3k^3}{\alpha^2}\cdot\frac{1}{k^3} - \frac{k^4}{\alpha^2}\cdot\frac{1}{k^4} > \frac{2}{5\alpha},$$

as $\alpha > 100$ by Assumption 2. ∎ Claim 5.4.8

The second condition is that no other edge of $\overline{G}_{\text{easy}}$ should appear between (V_i, V_j) (conditioned on e already appearing between V_i and V_j). Since $\overline{G}_{\text{easy}}$ has $\approx \text{opt} \cdot \text{poly}\log(n)$ edges only and each edge appears between (V_i, V_j) specifically with probability only $1/k^2 \approx \alpha^2/\text{opt}^2$, the probability that another edge appears between V_i and V_j is only $o(1)$. Formally,

Claim 5.4.9 (Condition (ii) of weakly-represented)**.**

$$\Pr\big(\text{another edge of } \overline{G}_{\text{easy}} \text{ appear between } (V_i,V_j) \mid e=(u,v) \in (V_i,V_j)\big) = o(1).$$

Proof. We partition the edges of $\overline{G}_{\text{easy}}$ into two parts: the edges $E_1 := E_1(\overline{G}_{\text{easy}}, u, v)$ that are incident on either u or v, and the remaining edges $E_2 := E_2(\overline{G}_{\text{easy}}, u, v)$. By union bound,

$$\begin{aligned}\text{LHS of Claim 5.4.9} &\leqslant \sum_{f \in E_1} \Pr(f \in (V_i, V_j) \mid e \in (V_i, V_j)) + \sum_{f \in E_2} \Pr(f \in (V_i, V_j) \mid e \in (V_i, V_j)) \\ &= |E_1| \cdot \frac{1}{k} + |E_2| \cdot \frac{1}{k^2},\end{aligned}$$

where each term of the second inequality is because: for the first term, assuming $f_1 = (u, w)$ for $w \neq v$ (the other case is symmetric), we need to have $h_j(w) = h_j(v)$ which happens with probability $1/k$ as $h_j(\cdot)$ is (> 2)-wise independent; and for the second term, assuming $f = (w, z)$ for $w \neq u$ and $z \neq v$ we need to have $h_i(w) = h_i(u)$ and $h_j(z) = h_j(v)$ which only happens with probability $1/k^2$ as both $h_i(\cdot)$ and $h_j(\cdot)$ are (> 2)-wise independent, and also independent of each other.

Moreover, as $e \in M^\star$, we have that $|E_1| \leqslant \tilde{b} = n^{\delta/4}$ (Eq (5.3)) and by sparsify-case property of Lemma 5.3.2, we have $|E_2| < 20\,\text{opt} \cdot \log^4 n$. Since $k = (10\text{opt}/\alpha)$, we have,

$$\text{LHS of Claim 5.4.9} \leqslant n^{\delta/4} \cdot \frac{\alpha}{10\,\text{opt}} + (20\,\text{opt} \cdot \log^4 n) \cdot \frac{\alpha^2}{100\,\text{opt}^2} = o(1),$$

as $\text{opt} \geqslant \alpha^2 \cdot n^\delta$ by Assumption 1. ∎ Claim 5.4.9

Finally, the last condition is that both V_i and V_j should be clean, namely, there is no vertex of M_{easy} inside either of them (again, conditioned on e already appearing between (V_i, V_j)). There are at most $(\text{opt}/4\alpha)$ vertices in M_{easy} and each one appear in either group with probability $1/k \approx \alpha/\text{opt}$, thus we can bound the probability that neither group has any vertex of M_{easy} by some small constant. Formally,

Claim 5.4.10 (Condition *(iii)* of weakly-represented)**.**

$$\Pr(\text{one of } V_i \text{ or } V_j \text{ is not clean} \mid e = (u, v) \in (V_i, V_j)) < \frac{1}{20}.$$

Proof. Recall that M_{easy} has less than $(\text{opt}/4\alpha)$ vertices. By union bound, we thus have,

$$\text{LHS of Claim 5.4.10} \leqslant \sum_{w \in V(M_{\text{easy}})} \Big(\Pr(h_i(w) = 1 \mid e \in (V_i, V_j)) + \Pr(h_j(w) = 1 \mid e \in (V_i, V_j)) \Big)$$

$$= |V(M_{\text{easy}})| \cdot \left(\frac{1}{k} + \frac{1}{k}\right)$$
(as both of $h_i(\cdot)$ and $h_j(\cdot)$ or (>2)-wise independent)
$$< \frac{\text{opt}}{4\alpha} \cdot \frac{\alpha}{5 \cdot \text{opt}} = \frac{1}{20},$$

by the choice of $k = 10\text{opt}/\alpha$. ∎ Claim 5.4.10

We can now conclude the proof of Lemma 5.4.7.

Proof of Lemma 5.4.7. An edge $e \in M^\star$ is weakly-represented iff it satisfies all the conditions (i) to (iii) of Definition 5.4.4. Claim 5.4.8 lower bounds the probability that e satisfies condition (i). Conditioned on this event, Claims 5.4.9 and 5.4.10 each upper bound the probability that e does not satisfy conditions (ii) or (iii), respectively. Thus,

$$\Pr(e \text{ is weakly-represented}) \geqslant \frac{2}{5\alpha} \cdot (1 - o(1) - \frac{1}{20}) > \frac{1}{3\alpha},$$

concluding the proof. ∎ Lemma 5.4.7

Let WR be a random variable for the number of weakly-represented edges. By Claim 5.4.6 and Lemma 5.4.7,

$$\mathbb{E}[WR] = |M^\star| \cdot \Pr(\text{weakly-represented}) \geqslant \frac{2\,\text{opt}}{3} \cdot \frac{1}{3\alpha} \geqslant \frac{\text{opt}}{6\alpha}. \tag{5.4}$$

While there is a degree of correlation between different edges of $M^\star$ being weakly-represented, it is not too much and thus we can prove WR is also concentrated using a careful argument. We first need to bound the number of weakly-represented edges that can appear inside any group.

Claim 5.4.11. *For any group $V_i \in \mathcal{V}$,*

$$\Pr\big(\textit{more than } (\log^2 n) \textit{ weakly-represented edges appear inside } V_i\big) \ll 1/\text{poly}(n).$$

Proof. As each weakly-represented edge belong to $M^\star$, we simply upper bound the number of edges of $M^\star$ that appear inside V_i. For any edge $e \in M^\star$, define an indicator random variable $X_e \in \{0, 1\}$ which is 1 iff $e \in V_i$. For $X_e = 1$ to happen at least one endpoint of e should be in V_i and at least one endpoint of e should be in one of k/α neighboring groups

of V_i (this is an "upper bound" because these endpoints of e should be different, but we are only interested in upper bounding the probability of $X_e = 1$). As such,

$$\begin{aligned}\mathbb{E}[X_e] &\leqslant \Pr(\text{one end of } e \text{ is in } V_i) \cdot \Pr(\text{one end of } e \text{ is in a neighbor of } V_i) \\ &\qquad \text{(by the independence of choice of vertices in different groups)} \\ &\leqslant \frac{2}{k} \cdot \frac{k}{\alpha} \cdot \frac{2}{k} \leqslant \frac{1}{\mathrm{opt}}.\end{aligned}$$

(by the choice of $k = 10\mathrm{opt}/\alpha$ and as each vertex belongs to a group with probability $1/k$)

Define $X := \sum_{e \in M^\star} X_e$ as the number of edges of $M^\star$ appearing inside V_i. As such, $\mathbb{E}[X] \leqslant 1$. Moreover, $M^\star$ is a matching (thus vertex-disjoint edges), choice of vertices inside each V_i is $(\log^2 n)$-wise independent, and choice of different V_i's are independent. Hence, the set of variables $\{X_e\}_{e \in M^\star}$ are $(\log^2 n)$-wise independent. By concentration results for sum of $(\log^2 n)$-wise independent random variables (Proposition 2.4.8),

$$\Pr\big(X > (\log^2 n)\big) \leqslant \Pr\big(X > (\log^2 n) \cdot \mathbb{E}[X]\big) \leqslant \exp\left(-\frac{(\log^2 n)}{6}\right) \ll 1/\mathrm{poly}(n).$$

A union bound over all groups concludes the proof. ∎ Claim 5.4.11

We now prove that the random variable WR is concentrated.

Claim 5.4.12 (Number of weakly-represented edges is concentrated)**.**

$$\Pr\left(|WR - \mathbb{E}[WR]| > \frac{\mathrm{opt}}{42 \cdot \alpha}\right) \ll 1/\mathrm{poly}(n).$$

Proof. WR is the sum of $|M^\star|$ random variables, each determining whether a given edge in $M^\star$ is weakly-represented or not. These random variables are *not* independent because an edge between V_i and V_j being weakly-represented restricts other edges between V_i and V_j from being weakly-represented. Thus, we cannot directly apply the Chernoff bound. Instead, we are going to use McDiarmid's inequality (Proposition 2.4.5).

Define random variables $Z_1, \ldots, Z_k$ as the choice of vertices in each group in $\mathcal{V}$, i.e., each $Z_i \subseteq V$ and is equal to the set of vertices in V_i. The choice of random variables $\{Z_i\}$ are independent of each other (as they are decided by different hash functions $\{h_i(\cdot)\}$). The value of random variable WR is a deterministic function of $Z_1, \ldots, Z_k$ so we can set $WR = f(Z_1, \ldots, Z_k)$ for some function f. To apply Proposition 2.4.5, we need to have that $f(\cdot)$ is Lipschitz (for some relatively small parameter), but this is not the case in general. We fix this in the following.

Define another function $g(Z_1, \ldots, Z_k)$ as follows. Let g count the number of weakly-represented edges defined by the choices of $Z_1, \ldots, Z_k$ that appear between groups with at most $(\log^2 n)$ other weakly-represented edges appearing inside them. By Claim 5.4.11, we have that,

$$\Pr(f \neq g) \ll 1/\mathrm{poly}(n) \quad \text{and thus} \quad \mathbb{E}[g] \geqslant \mathbb{E}[f] - 1.$$

As such, we can prove a concentration for g instead of f and obtain the result for f as well. We now prove that g is $(2\log^2 n)$-Lipschitz which allows us to prove its concentration. Suppose we change the realization of a single variable Z_i from a set of vertices U_i to U_i'. Then, the following may happen:

- Both U_i and U_i' have less than $(\log^2 n)$ weakly-represented edges appearing inside them: Thus, the change of U_i to U_i' can only change the value of g by at most $(2\log^2 n)$.
- U_i has more and U_i' has less than $(\log^2 n)$ weakly-represented edges appearing inside them: In g, none of the weakly-represented edges incident on U_i were counted. In U_i', at most $(\log^2 n)$ new edges will be counted toward g. This changes the value of g by at most $(\log^2 n)$.
- U_i has less and U_i' has more than $(\log^2 n)$ weakly-represented edges appearing inside them: At most all the $(\log^2 n)$ weakly-represented edges incident on U_i are going to be not counted toward g when switching to U_i', thus changing the value of g by at most $(\log^2 n)$.
- Both U_i and U_i' have more than $(\log^2 n)$ weakly-represented edges appearing inside them: Neither contributed any value to g so value of g remains the same.

Thus, g is $(2\log^2 n)$-Lipschitz. Given this, we can apply McDiarmid's inequality (Proposition 2.4.5) and get,

$$\Pr\left(|g - \mathbb{E}[g]| > \frac{\mathrm{opt}}{42 \cdot \alpha}\right) \leqslant 2 \cdot \exp\left(-\frac{2 \cdot \mathrm{opt}^2}{42^2 \cdot \alpha^2 \cdot k \cdot (2\log^2 n)^2}\right) \ll 1/\mathrm{poly}(n).$$

(as by Assumption 2 $\mathrm{opt} \geqslant \alpha^2 \cdot n^\delta$)

This concludes the proof. ∎ Claim 5.4.12

By Claim 5.4.12 and Eq (5.4), we have that with high probability

$$WR > \frac{\mathrm{opt}}{7 \cdot \alpha}. \tag{5.5}$$

Let us now bound the number of strongly-represented edges, denoted by SR. In the following lemma, we bound the number of strongly-represented edges in an indirect way by bounding how many edges among weakly-represented edges can no longer be strongly-represented.

Lemma 5.4.13. *With probability* $1 - n^{-\delta/6}$*, we have* $SR > WR - \dfrac{\text{opt}}{56 \cdot \alpha}$.

We need the following claim that bounds the probability that a group is expanding. We will then simply subtract the edges of all expanding groups from the weakly-represented edges to obtain the number of strongly-represented one. This probability should be small because $\overline{G}_{\text{easy}}$ only has $\approx \text{opt} \cdot \text{poly}\log(n)$ edges and each edge appear inside a group with probability roughly $\approx 1/\text{opt}$. Formally,

Claim 5.4.14. *For any group* V_i, $\Pr(V_i \text{ is expanding}) < n^{-\delta/5}$.

Proof. For V_i to be expanding, at least $\tilde{b}$ edges of $\overline{G}_{\text{easy}}$ should appear inside V_i. Let $E_1 := E(\overline{G}_{\text{easy}})$ denote the edges in $\overline{G}_{\text{easy}}$. We have that size of E_1 is at most $(20\text{opt} \cdot \log^4 n)$ by sparsify-case of Lemma 5.3.2.

For any $e \in E_1$, let $X_e \in \{0, 1\}$ be an indicator random variable which is 1 iff e appears inside V_i. As we proved in Claim 5.4.11,

$$\mathbb{E}[X_e] < \frac{1}{\text{opt}}.$$

Let $X := \sum_{e \in E_1} X_e$ denote the number of edges of E_1 that appear inside V_i. Thus, by Markov bound (and since $\tilde{b} = n^{\delta/4}$),

$$\Pr\left(X > \tilde{b}\right) \leqslant \frac{\mathbb{E}[X]}{\tilde{b}} \leqslant \frac{1}{\tilde{b}} \cdot |E_1| \cdot \frac{1}{\text{opt}} = n^{-\delta/4} \cdot (20\text{opt} \cdot \log^4 n) \cdot \frac{1}{\text{opt}} \ll n^{-\delta/5},$$

concluding the proof. ∎ Claim 5.4.14

Proof of Lemma 5.4.13. Let X denote the number of expanding groups. By Claim 5.4.14, combined with a Markov bound (and linearity of expectation),

$$\Pr\left(X > \frac{\text{opt}}{\alpha \cdot 56 \cdot (\log^2 n)}\right) \leqslant (56 \log^2 n) \cdot \frac{\alpha}{\text{opt}} \cdot k \cdot n^{-\delta/5} \ll n^{-\delta/6},$$

as $k = (10 \cdot \text{opt}/\alpha)$. Additionally, by Claim 5.4.11, at most $(\log^2 n)$ weakly-represented edges appear in each group with high probability. Thus, with probability at least $1 - n^{-\delta/6}$, the total number of weakly-represented edges incident on expanding groups is at most

$$\frac{\text{opt}}{56 \cdot \alpha \cdot \log^2 n} \cdot \log^2 n = \frac{\text{opt}}{56 \cdot \alpha}.$$

All remaining weakly-represented edges will also be strongly-represented, proving the lemma. ∎ Lemma 5.4.13

We can now bound the number of strongly represented edges. Lemma 5.4.5 follows by combining Eq (5.5) with Lemma 5.4.13. We can conclude that,

$$\Pr\left(\text{number of strongly-represented edges} \geqslant \frac{\text{opt}}{8 \cdot \alpha}\right) \geqslant 1 - n^{-\delta/6}. \tag{5.6}$$

5.4.2 The Recovery Algorithm

We now show how to recover a large matching from the sketch computed by Algorithm 9. The idea is to find all strongly-represented edges (or rather a superset of them). This is enough because the number of strongly-represented edges is at least $\text{opt}/8\alpha$ with high enough probability (Lemma 5.4.5). To do so, we first remove all groups that have a vertex from M_{easy} inside them. Next, we run a "weak tester" using sketches for `NE-Tester` to essentially remove all expanding sets (this step is done slightly differently in the algorithm). Finally, we use `SN-Recovery` to recover the neighborhood of each group inside $\overline{G}_{\text{easy}}$ by setting the T-set of the sketches in the recovery as vertices matched by M_{easy}. Then, whenever between two groups we only recovered a single pair of vertices, we consider this pair as an edge and store them[10]. At the end, we compute a maximum matching among the stored edges.

Algorithm 10. The recovery algorithm of Lemma 5.4.1.

Input: Groups $\mathcal{V}$ and sketches computed by Algorithm 9 and a matching M_{easy} (of Lemma 5.3.2).

[10] `SN-Recovery` can find the neighbor vertex v of a group V_i inside another group V_j; however, it cannot recover an edge because it does not specify the endpoint of neighbor v inside V_i. This is fixed by the process mentioned above by also finding a unique neighbor inside V_i from the `SN-Recovery` run on V_j instead

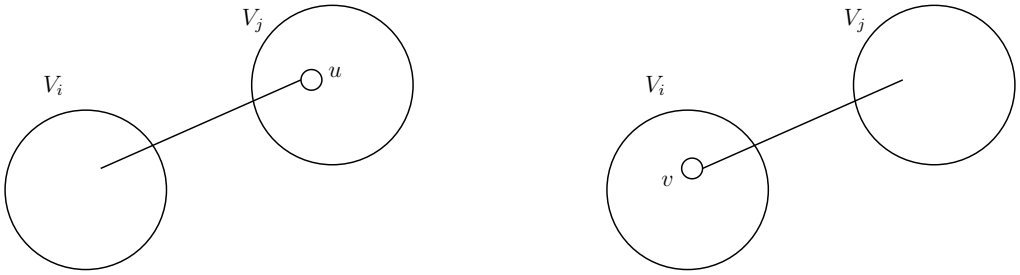
Output: A matching M_{hard} in G.

(*i*) For every group $V_i \in \mathcal{V}$, define T_i as the vertices in the graph $G(V_i)$ (defined in Algorithm 9) that also appear in M_{easy}. Run the following tests:

- M_{easy}**-test**: if V_i has any vertex of M_{easy} inside it, **remove** V_i;
- **Expanding-test**: Run the recovery algorithm of `NE-Tester`$(G(V_i), V_i)$ with the set $T = T_i$ to test if V_i has at most $\tilde{b}$ neighbors or at least $2\tilde{b}$ neighbors out of T_i in $G(V_i)$: if $2\tilde{b}$, **remove** V_i.

(*ii*) For any remaining group V_i, run the recovery algorithm `SN-Recovery`$(G(V_i), V_i)$ with set $T = T_i$ to recover $NR(V_i)$ in the graph $G(V_i)$.

(*iii*) Define the following **recovered graph** H_{rec} on vertices V. For any two remaining groups V_i, V_j, if sizes of both $NR(V_i) \cap V_j$ and $NR(V_j) \cap V_i$ is 1, add the edge (u, v) to H where u and v are the unique vertices in the aforementioned sets. Return M_{hard} as a maximum matching of H.



(a) This figure shows the group V_i and $NR(V_i)$ has only one element u in V_j.

(b) This figure shows the group V_j and $NR(V_j)$ has only one element v in V_i.

Figure 5.4: Illustration of two groups V_i and V_j that pass the M_{easy}-test and Expanding-test. We have that $NR(V_i)$ has only one element u in V_j and $NR(V_j)$ has only one element v in V_i. This means that (u, v) must be an edge in G.

Figure 5.4 shows which edges are added to H. Our goal now is to show that with high probability H_{rec} contains all strongly-represented edges and moreover it does not contain any edge that is not part of G. Putting these two together with Lemma 5.4.5 then finalizes the proof.

The first step is to show that for both `NE-Tester` and `SN-Recovery` sketches run by

the algorithm, the promise on the input is satisfied. We first prove that the set T_i satisfies $|T_i| \leqslant a$ for all $i \in [k]$ (as required by both algorithms with given parameter a).

Claim 5.4.15. *With high probability, for every $i \in [k]$, size of T_i is at most a.*

Proof. For any $v \in V(M_{\text{easy}})$, define an indicator random variable $X_v \in \{0,1\}$ which is 1 iff v is in T_i, i.e., there exists an index j such that $h_j(v) = 1$ and $F(i,j) = 1$. Given there are exactly (k/α) choices for $j \in [k]$ with $F(i,j) = 1$, we have that by union bound,

$$\mathbb{E}[X_v] \leqslant \frac{k}{\alpha} \cdot \frac{1}{k} = \frac{1}{\alpha},$$

as each $h_j(\cdot)$ is uniform over $[k]$. Let $X := \sum_{v \in V(M_{\text{easy}})} X_v$ denote the size of T_i. As there are at most $(\text{opt}/4\alpha)$ vertices in M_{easy}, we get

$$\mathbb{E}[X] \leqslant |V(M_{\text{easy}})| \cdot \frac{1}{\alpha} \leqslant \frac{\text{opt}}{4\alpha^2}.$$

Finally, note that random variables $\{X_v\}$ are only correlated through the choice of $(\log^2 n)$-wise independent hash functions $\{h_j(\cdot)\}$ (which are themselves independent for different h_j's). Thus, by concentration results for sum of $(\log^2 n)$-wise independent random variables (Proposition 2.4.8),

$$\Pr(X > a) \leqslant \Pr(X > 8 \cdot \mathbb{E}[X]) \leqslant \exp\bigl(-(\log^2 n)/2\bigr) \ll 1/\text{poly}(n).$$

A union bound over all groups concludes the proof. ∎ Claim 5.4.15

The above claim along with $a \geqslant 16\tilde{b}$ (by Assumption 2) is enough for running `NE-Tester`. We now show that the guarantees for `SN-Recovery` are also satisfied. This first requires proving that $N(V_i) - T_i$, for each V_i that is not removed by the algorithm, has size at most $2\tilde{b}$ (here, and throughout the rest of the analysis $N(V_i)$ is in the graph $G(V_i)$).

Claim 5.4.16. *With high probability, for every remaining group V_i, size of $N(V_i) - T_i$ is at most $2\tilde{b}$.*

Proof. Given that the promise to `NE-Tester`$(G(V_i), V_i)$ is satisfied by Claim 5.4.15, with high probability, `NE-Tester` is going to only output $\tilde{b}$-case for a set V_i when $N(V_i) - T_i$ has size less than $2\tilde{b}$ by Lemma 5.2.9. Thus, any group not removed satisfies $|N(V_i) - T_i| \leqslant 2\tilde{b}$ as desired. ∎ Claim 5.4.16

Finally, we also need to prove that for any vertex $v \in N(V_i) - T_i$, size of $V_i \cap N(v)$ is at most c (again $N(V_i)$ in the graph $G(V_i)$). This is done in a rather indirect way in the following claim.

Claim 5.4.17. *With high probability, for every remaining group V_i and any of its neighbor group V_j, size of $N(V_i) \cap V_j$ is at most c.*

Proof. We condition on the group V_i remaining after the tests. By Claim 5.4.16, we have that there are at most $2\tilde{b}$ vertices in $N(V_i) - T_i$. Let O_i denote the set of these vertices.

Note that the randomness of this conditioning is only a function of the graph $G(V_i)$ and vertices V_i. Recall that the definition of graph $G(V_i)$ was the following set of edges $\{(u, v) \in E \mid u \in V_i, v \in V_j, F(i, j) = 1\}$. This definition depends on the *union* of the sets $\{V_j\}_{F(i,j)=1}$ but *not the partitioning* of vertices into different groups. This is not true if multiple copies of a vertex exist in O_i because if one copy belongs to a group then the others cannot belong to the same group. Assuming that multiple copies of a vertex could exist in a group only increases the size of $O_i \cap V_j$ which is okay for us since we only need an upper bound. Therefore, we can assume that each vertex of O_i has an *equal* probability of belonging to each of these (k/α) groups.

$$\begin{aligned}
\Pr(|O_i \cap V_j| > c) &\leqslant \binom{|O_i|}{c} \cdot \left(\frac{\alpha}{k}\right)^c \leqslant \left(\frac{2\tilde{b} \cdot \alpha}{k}\right)^c && \text{(as size } O_i \text{ is at most } 2\tilde{b}) \\
&\leqslant \left(\frac{2 \cdot n^{\delta/4} \cdot \alpha^2}{10 \cdot \mathrm{opt}}\right)^{30/\delta} \\
&\qquad \text{(by the choice of } \tilde{b} = n^{\delta/4}, c = (30/\delta), \text{ and } k = (10\mathrm{opt}/\alpha)) \\
&< \left(n^{-\delta/2}\right)^{30/\delta} = n^{-15}. && \text{(by Assumption 2 } \mathrm{opt} \geqslant \alpha^2 \cdot n^{\delta})
\end{aligned}$$

A union bound over all choices for V_i concludes the proof. ∎ Claim 5.4.17

We now show that Claim 5.4.17 implies that for every $v \in N(V_i) - T_i$, size of $V_i \cap N(v)$ is at most c, thus satisfying the guarantee of `SN-Recovery`. Suppose the event of Claim 5.4.17 happens for all remaining groups. This means that among the remaining groups, between every pair of neighbor groups V_i and V_j, there can be at most c vertices in V_i that have an edge to V_j. Naturally, any vertex in union of all V_j's can then also only have c neighbors in V_i. Since all of $N(V_i) - T_i$ is now a subset of these remaining V_j's, we get the desired guarantee. To conclude, by Claims 5.4.15 to 5.4.17 we established that the guarantees required by `NE-Tester` and `SN-Recovery` are all satisfied.

We now show how the algorithm can recovers strongly-represented edges. The first step is to show that the endpoint-groups of strongly-represented edges will not be removed by Algorithm 10 with high probability.

Claim 5.4.18. *Suppose e is a strongly-represented edge by groups (V_i, V_j). Then, with high probability, neither of V_i nor V_j will be removed by Algorithm 10.*

Proof. Since e is strongly-represented, we know both V_i and V_j are clean (thus will not be removed by M_{easy}-test) and are non-expanding, thus each have at most $\tilde{b}$ other edges of $\overline{G}_{\text{easy}}$ appearing inside them. Thus, these groups will also pass the expanding-test, and so they will not be removed as long as `NE-Tester` is working correctly which happens with high probability. ∎ Claim 5.4.18

Lemma 5.4.19. *With high probability, H_{rec} is a subgraph of G and contains all strongly-represented edges.*

Proof. We condition on the high probability events of Claims 5.4.15 to 5.4.18 and correctness of `NE-Tester` and `SN-Recovery`.

Firstly, consider an edge (u, v) added to H_{rec} and assume $u \in V_i$ and $v \in V_j$. For (u, v) to be included in H_{rec}, we need to have $N(V_i) \cap V_j = \{v\}$ and $N(V_j) \cap V_i = \{u\}$. This means there is an edge (u, v) in G as well, proving the second part of the lemma.

For the first part, consider any strongly-represented edge (u, v) and again assume $u \in V_i$ and $v \in V_j$ (condition (i) of weakly-represented edges). These groups will not be removed as we conditioned on the event of Claim 5.4.18. Moreover, there is no other edge of $\overline{G}_{\text{easy}}$ between V_i and V_j (condition (ii) of weakly-represented edges) and since V_i and V_j do not have any vertex of M_{easy} (condition (iii) of weakly-represented edges), we have that (u, v) is the unique edge between V_i and V_j. By the argument above, this means (u, v) will be added to H_{rec}. ∎ Lemma 5.4.19

We can now conclude the proof of Lemma 5.4.1. Firstly, by Lemma 5.4.19, with high probability the algorithm does not make an error. Moreover, by Lemma 5.4.5, with probability at least $1 - n^{-\delta/6}$, there are at least $(\text{opt}/8\alpha)$ strongly-represented edges. Since these edges are coming from a matching $M^\star$ and by Lemma 5.4.19 we get all of those in H_{rec}, the output matching M_{hard} is going to have size at least $(\text{opt}/8\alpha)$. As we already established the bound on the space in Lemma 5.4.2, this concludes the proof.

5.5 Match-Or-Sparsify Lemma

We prove Lemma 5.3.2, restated below, in this section. Informally speaking, given a graph G, this lemma gives an algorithm that either finds a large matching in G or identifies a sparse induced subgraph of G that contains a large matching.

Lemma (Re-statement of Lemma 5.3.2)**.** *There is a linear sketch that given any graph $G = (V, E)$ specified via* $\mathbf{vec}(E)$*, uses $O(\text{opt}^2/\alpha^3)$ bits of space and with high probability outputs a matching M_{easy} that satisfies at least one of the following conditions:*

- ***Match-case:*** *The matching M_{easy} has at least $(\text{opt}/8\alpha)$ edges;*
- ***Sparsify-case:*** *The induced subgraph of G on vertices not matched by M_{easy}, denoted by $\overline{G}_{\text{easy}}$, has at most $(20\,\text{opt} \cdot \log^4 n)$ edges and a matching of size at least $3\,\text{opt}/4$.*

The algorithm in Lemma 5.3.2 samples $\approx \text{opt}^2/(\alpha \cdot \log n)^3$ edges from the graph using a non-uniform distribution as follows: for each sample, we first pick $\approx \text{opt}/\alpha$ vertices S uniformly at random and then use `NE-Sampler` to sample an edge from S to a vertex of $N(S)$ chosen uniformly at random. Given the bound of $O(\log^3 n)$ bits on the size of sketches for `NE-Sampler`, the total space of the algorithm can be bounded by $O(\text{opt}^2/\alpha^3)$ bits. In the recovery phase then, we compute a greedy matching over these sampled edges and return it as M_{easy}. Formally, the algorithm is as follows.

Algorithm 11. The algorithm of Match-Or-Sparsify Lemma (Lemma 5.3.2).

Input: A graph $G = (V, E)$ specified via $\mathbf{vec}(E)$; **Output:** A matching M_{easy} in G.

Parameters: Let $k := \text{opt}/\alpha$ and $s := \text{opt}^2/(\alpha \cdot \log n)^3$.

Sketching matrix:

1. For $i = 1$ to $(2s)$ steps[a]:
   (a) Sample a pair-wise independent hash function $h_i : V \to [k]$ and set $V_i := \{v \in V \mid h_i(v) = 1\}$.
   (b) Let $\Phi(V_i)$ be the sketching matrix of `NE-Sampler`(G, V_i).
2. Return $\Phi := [\Phi(V_1); \cdots; \Phi(V_{2s})]$ as the sketching matrix.

Recovery algorithm:

1. For all $i \in [2s]$, run the recovery algorithm of `NE-Sampler`(G, V_i) using $\Phi(V_i)$ and $\Phi(V_i) \cdot \mathbf{vec}(E)$ to get an output edge e_i (we write $e_i = \perp$ if the sampler outputs FAIL).

2. Let $M_{\text{easy}} \leftarrow \emptyset$ initially. For $i = 1$ to $2s$ steps: greedily include e_i in M_{easy} whenever $e_i \neq \perp$ and both its endpoints are unmatched by M_{easy}.

[a] We partition the steps into two **batches** of s steps each in the analysis, hence the use of $2s$ for the number of steps.

Note that it is equivalent to think of the edges being recovered one by one and fed to the greedy matching algorithm. We will use this in our analysis. We first bound the space complexity of this algorithm.

Lemma 5.5.1. *Algorithm 11 uses $O(\text{opt}^2/\alpha^3)$ bits of space with high probability.*

Proof. In each step, Line 1.a requires storing a pair-wise independent hash function which needs $O(\log n)$ bits of space by Proposition 2.4.7. Line 1.b requires storing an `NE-Sampler` which needs $O(\log^3 n)$ bits by Lemma 5.2.1. There are $2s = O(\text{opt}^2/(\alpha \cdot \log n)^3)$ steps, so the total space $O(\text{opt}^2/\alpha^3)$ bits. ∎

We now prove that the matching M_{easy} output by Algorithm 11 satisfies the guarantees of Lemma 5.3.2. To continue, we need some notation.

Notation. For any $i \in [2s]$, let M_i be the set of edges included in M_{easy} in the first $i-1$ steps of the recovery, i.e., from $\{e_j\}_{j=1}^{i-1}$, and $\overline{G}_i$ to be the subgraph of G induced on *unmatched* vertices of M_i. We use $\deg_i(v)$ to denote the degree of each vertex in $\overline{G}_i$ to other vertices in $\overline{G}_i$. We partition vertices of $\overline{G}_i$ based on their degrees in $\overline{G}_i$ into **low-**, **medium-**, and **high-degree** as follows:

$$\text{Low}_i := \left\{v : \deg_i(v) < (\alpha \log^3 n)\right\}, \quad \text{Med}_i := \left\{v : (\alpha \log^3 n) \leqslant \deg_i(v) < (\frac{\text{opt}}{8\alpha})\right\},$$

$$\text{High}_i := \left\{v : \deg_i(v) \geqslant \frac{\text{opt}}{8\alpha}\right\}.$$

We define the following two events:

- $\mathcal{E}_{\mathrm{M}}(i)$: the matching M_i has less than $(\mathrm{opt}/8\alpha)$ edges (i.e., matching-case not happened);
- $\mathcal{E}_{\mathrm{S}}(i)$: the subgraph $\overline{G}_i$ has more than $(20\,\mathrm{opt}\cdot\log^4 n)$ edges (i.e., sparsify-case not happened).

Finally, we say that a choice of V_i in step $i \in [2s]$ is **clean** if V_i does not contain any matched vertices of M_i.

We start by proving that if for some $i \in [2s]$ at least one of these events do not happen, then Algorithm 11 succeeds in outputting the desired matching of Lemma 5.3.2. The proof is straightforward.

Claim 5.5.2. *Suppose for some $i \in [2s]$, either of $\mathcal{E}_{\mathrm{M}}(i)$ or $\mathcal{E}_{\mathrm{S}}(i)$ does not happen; then, M_{easy} of Algorithm 11 satisfies the guarantees of Lemma 5.3.2.*

Proof. Suppose first that $\mathcal{E}_{\mathrm{M}}(i)$ does not happen. This means M_i has size at least $(\mathrm{opt}/8\alpha)$ and by the greedy choice of M_{easy}, we have $|M_{\mathrm{easy}}| \geqslant |M_i| \geqslant (\mathrm{opt}/8\alpha)$, satisfying the match-case condition.

Now suppose that $\mathcal{E}_{\mathrm{S}}(i)$ does not happen. Since the number of edges of $\overline{G}_{\mathrm{easy}}$ can only be smaller than that of $\overline{G}_i$, we have that $\overline{G}_{\mathrm{easy}}$ also only has $(20\,\mathrm{opt}\cdot\log^4 n)$ edges. We can also assume that size of M_{easy} is at most $(\mathrm{opt}/8\alpha)$ as otherwise we will be done by the matching-case. This means that at most $(\mathrm{opt}/4\alpha)$ vertices of any maximum matching of G are incident on M_{easy}, thus $\overline{G}_{\mathrm{easy}}$ still contains a matching of size at least $\mathrm{opt} - (\mathrm{opt}/4\alpha) > 3\,\mathrm{opt}/4$, satisfying the sparsify-case condition. ∎

The goal at this point is to show that with high probability, for some $i \in [2s]$, one of the events $\mathcal{E}_{\mathrm{M}}(i)$ or $\mathcal{E}_{\mathrm{S}}(i)$ is *not* going to happen. In order to do so, we partition the steps of the algorithm into two **batches** of size s each and analyze each one separately as follows:

- **First batch:** We first show that as long as $\mathcal{E}_{\mathrm{M}}(i)$ and $\mathcal{E}_{\mathrm{S}}(i)$ happen for all $i \in [s]$, with high probability, the set High_{s+1} (and thus High_j for all $j \in (s, 2s]$) will be empty for the second batch (a technical condition needed for our variance reduction ideas in the next part). Formally,

  Lemma 5.5.3. *With high probability, either at least one of $\mathcal{E}_{\mathrm{M}}(i)$ and $\mathcal{E}_{\mathrm{S}}(i)$ does not happen for some step $i \in [s]$ or High_{s+1} will be empty.*

- **Second batch:** We then show that whenever both $\mathcal{E}_{\mathrm{M}}(i)$ and $\mathcal{E}_{\mathrm{S}}(i)$ happen in a step $i \in (s, 2s]$, there will be a probability of $\approx k/s$ in increasing the size of M_i by one in this step (this is the main part of the argument). Given that we repeat this process for s steps also, this allows us to argue M_{easy} will eventually become of size $\approx k = \text{opt}/\alpha$, thus satisfying the matching-case condition (or one of the events happen along the way, and we can use Claim 5.5.2 instead). Formally,

  Lemma 5.5.4. *Assuming* High_{s+1} *is empty, with high probability, at least one of the events* $\mathcal{E}_{\mathrm{M}}(i)$ *or* $\mathcal{E}_{\mathrm{S}}(i)$ *does not happen for some* $i \in (s : 2s]$.

Lemma 5.3.2 then follows immediately from these two lemmas combined with Lemma 5.5.1 and Claim 5.5.2.

Before we get to the proofs of these lemmas, we make the following important remark.

Remark 5.5.5. The actions of Algorithm 11 are clearly *not* independent across different steps (in the recovery phase). However, in our upcoming probability analysis in each step $i \in [2s]$ we fix the randomness of all prior steps conditioned on that events $\mathcal{E}_{\mathrm{M}}(i)$ and $\mathcal{E}_{\mathrm{S}}(i)$, and use *only* the randomness of the choice of (V_i, e_i) in this step. This randomness is independent of prior steps. As such, in the following, **all our probability calculations in a step i are conditioned on randomness of prior steps and events $\mathcal{E}_{\mathrm{M}}(i)$ and $\mathcal{E}_{\mathrm{S}}(i)$**, without writing it explicitly each time. These probability calculations may not necessarily remain correct when either of these events do not happen, but we will be done by Claim 5.5.2 in those cases anyway.

We use the following simple helper claim in the subsequent proofs (this claim would have been trivial had h_i was a truly independent hash function instead of a pairwise-independent one).

Claim 5.5.6. *Consider any step* $i \in [s]$ *and let* v *be any arbitrary vertex in* $\overline{G}_i$. *Then,*

$$\Pr_{V_i}\Big(v \in V_i \text{ and } V_i \text{ is clean}\Big) \geqslant \frac{3}{4k}.$$

Proof. Recall that there are at most $\text{opt}/4\alpha = k/4$ vertices matched by M_i. We have,

$$\Pr_{V_i}\Big(v \in V_i \text{ and } V_i \text{ is clean}\Big) = \Pr(h_i(v) = 1) \cdot \Pr(V_i \text{ is clean} \mid h_i(v) = 1)$$

$$(v \in V_i \text{ iff } h_i(v) = 1)$$

$$= \frac{1}{k} \cdot \Big(1 - \Pr(V_i \text{ is not clean} \mid h_i(v) = 1)\Big)$$
$$(\text{as } h_i(v) = 1 \text{ w.p. } 1/k)$$
$$\geqslant \frac{1}{k} \cdot \left(1 - \sum_{u \in V(M_i)} \Pr(h_i(u) = 1 \mid h_i(v) = 1)\right)$$
(by union bound and since V_i is not clean iff $h_i(u) = 1$ for some $u \in V(M_i)$)
$$= \frac{1}{k} \cdot \left(1 - \sum_{u \in V(M_i)} \frac{1}{k}\right)$$
($h_i(\cdot)$ is a pairwise-independent hash function)
$$= \frac{1}{k} \cdot \left(1 - \frac{k}{4} \cdot \frac{1}{k}\right),$$
(as $h_i(u) = 1$ w.p. $1/k$ and there are at most $k/4$ choices for matched vertices)

which is at least $3/4k$ as desired. ∎

5.5.1 First Batch: Proof of Lemma 5.5.3

Let v be any vertex in V and consider any step $i \in [s]$. If $\deg_i(v) < (\text{opt}/8\alpha)$, then v cannot be part of High_i and subsequently High_{s+1} since $\overline{G}_{s+1}$ is a subgraph of $\overline{G}_i$. In the following, we consider the case where $\deg_i(v) \geqslant (\text{opt}/8\alpha)$ and prove that there is a non-trivial chance of "progress" (to be defined later) in each step. We first bound the probability of the following useful event for our analysis.

Claim 5.5.7. *In step i, if $\deg_i(v) \geqslant \text{opt}/8\alpha$, we have $\Pr_{V_i}(v \in N(V_i) \text{ and } V_i \text{ is clean}) \geqslant \frac{1}{16}$.*

Proof. Let $d(v) := (\text{opt}/8\alpha)$ and $D(v)$ be a set of d arbitrary neighbors of v in $\overline{G}_i$. We know that v will be included in $N(V_i)$ if any of vertices in $D(v)$ is sampled in V_i. We have,

$$\Pr_{V_i}(v \in N(V_i) \text{ and } V_i \text{ is clean}) \geqslant \Pr(D(v) \cap V_i \neq \emptyset \text{ and } V_i \text{ is clean}) \qquad (D(v) \subseteq N(v))$$
$$\geqslant \sum_{u \in D(v)} \Pr(u \in V_i \text{ and } V_i \text{ is clean}) - \sum_{u \neq w \in D(v)} \Pr(u, w \in V_i)$$
(by inclusion-exclusion principle and bounding $\Pr(u, w \in V_i) \geqslant \Pr(u, w \in V_i \text{ and } V_i \text{ is clean})$)

$$> \frac{3d(v)}{4k} - \frac{d(v)^2}{k^2}$$

(by Claim 5.5.6 and as $h_i(\cdot)$ is a pair-wise independent hash function with range $[k]$)

$$= \frac{(\text{opt}/8\alpha)}{(\text{opt}/\alpha)} \cdot \left(\frac{3}{4} - \frac{(\text{opt}/8\alpha)}{(\text{opt}/\alpha)}\right),$$

(as $d(v) = \text{opt}/8\alpha$ and $k = \text{opt}/\alpha$)

which is at least $1/16$ as desired. ∎ Claim 5.5.7

Let us now condition on the choice of V_i and assume the event of Claim 5.5.7 has happened. We say that this step i is a **matching-step** if $N(V_i) > (\text{opt}/2\alpha)$; otherwise, we call this step a **vertex-step**. We argue that in a matching-step we have a constant probability of increasing the size of M_i by one and in a vertex-step we have a probability $\approx \alpha/\text{opt}$ of matching the vertex v and thus no longer including it in $\overline{G}_{i+1}$ and High_{i+1}. We formalize this in the following.

Claim 5.5.8. *Fix V_i and suppose step i is a matching-step and the event of Claim 5.5.7 has happened. Then,*

$$\Pr_{e_i}(e_i \in M_{i+1} \mid V_i) \geqslant \frac{1}{3}.$$

Proof. As $N(V_i)$ contains more than $(\text{opt}/2\alpha)$ vertices (as this is matching-step) while M_{easy} has at most $(\text{opt}/4\alpha)$ vertices (as $\mathcal{E}_{\text{M}}(i)$ has happened), we know that at least half the vertices in $N(V_i)$ are unmatched. Given that all of V_i is also unmatched, if $\texttt{NE-Sampler}(G, V_i)$ samples e_i to any of the unmatched vertices in $N(V_i)$, we can include e_i in M_{i+1} greedily. As the choice of $\texttt{NE-Sampler}(G, V_i)$ is uniform over $N(V_i)$, this event happens with probability at least $(1/2 - \delta_F) > 1/3$, as desired (since $\delta_F = 1/100$). ∎ Claim 5.5.8

Claim 5.5.9. *Fix V_i and suppose step i is a vertex-step and the event of Claim 5.5.7 has happened. Then,*

$$\Pr_{e_i}(v \in V(M_{i+1}) \mid V_i) \geqslant \frac{\alpha}{\text{opt}}.$$

Proof. We know $v \in N(V_i)$ and that size of $N(V_i)$ is at most $(\text{opt}/2\alpha)$. At the same time, since V_i is clean, if v is sampled as an endpoint of e_i by $\texttt{NE-Sampler}(G, V_i)$, the edge e_i will join the matching greedily and thus v will be matched. As the choice of $\texttt{NE-Sampler}(G, V_i)$ is uniform over $N(V_i)$ and $\delta_F = 1/100$,

$$\Pr_{e_i}(v \in V(M_{i+1}) \mid V_i) \geqslant (1 - \delta_F) \cdot \frac{1}{|N(V_i)|} > \frac{\alpha}{\text{opt}}.$$ ∎ Claim 5.5.9

We can now conclude the proof of Lemma 5.5.3 as follows. We have that at least half the steps are matching-steps or half of them are vertex-steps. We consider each case as follows.

When half the steps are matching-steps. In this case, each matching-step i increases size of M_i by one with probability at least $(1/48)$ by Claims 5.5.7 and 5.5.8. Thus,

$$\mathbb{E}\,|M_{s+1}| \geqslant (\frac{s}{2}) \cdot \frac{1}{48} = \frac{\mathrm{opt}^2}{(\alpha \cdot \log n)^3} \cdot \frac{1}{48} \gg \mathrm{opt}/\alpha,$$

given that $\mathrm{opt} \gg \alpha^2$ by Assumption 1. Moreover, the distribution of M_{s+1} statistically dominates sum of $(s/2)$ Bernoulli random variables with mean $(1/48)$. As such, by the Chernoff bound (Proposition 2.4.3),

$$\Pr(M_{s+1} < (\mathrm{opt}/8\alpha)) < \exp(-\mathrm{opt}/\alpha) \ll 1/\mathrm{poly}(n),$$

as $\mathrm{opt} \gg \alpha$ by Assumption 1. This implies that $\mathcal{E}_{\mathrm{M}}(s+1)$ happens, proving Lemma 5.5.3 in this case.

When half the steps are vertex-steps. In this case, each vertex-step i can independently match the vertex v with probability at least $(\alpha/16\,\mathrm{opt})$ by Claims 5.5.7 and 5.5.9. Thus,

$$\Pr\big(v \in \mathrm{High}_{s+1}\big) \leqslant (1-\frac{\alpha}{16\,\mathrm{opt}})^{s/2} \leqslant \exp\left(-\frac{\alpha}{16\,\mathrm{opt}} \cdot \frac{\mathrm{opt}^2}{2 \cdot (\alpha \cdot \log n)^3}\right) < \exp\left(-\frac{n^\delta}{32}\right) \ll 1/\mathrm{poly}(n),$$

where we use $\mathrm{opt} \geqslant \alpha^2 \cdot n^\delta$ by Assumption 2. Thus, with high probability v will not be part of High_{s+1}. A union bound over all the vertices $v \in V$ then ensures that High_{s+1} will be empty with high probability, thus proving Lemma 5.5.3 in this case too.

Remark: We note that the definition of matching-steps and vertex-steps are tailored to individual vertices in V; however, even if one vertex leads to having at least half of the steps as matching-steps, we can apply the argument of first part and conclude the proof. Thus, when applying the second part of the argument, we can assume that all vertices lead to half of the steps being vertex-steps, and so we can union bound over all of them.

5.5.2 Second Batch: Proof of Lemma 5.5.4

We now prove Lemma 5.5.4. In the following, we condition on the event that High_{s+1} (and High_i for every $i \in (s, 2s]$) is empty. Our goal is then to prove that at some step $i \in (s, 2s]$, one of the events $\mathcal{E}_{\mathrm{M}}(i)$ or $\mathcal{E}_{\mathrm{S}}(i)$ is not going to happen. The key to the proof of Lemma 5.5.4 (and Lemma 5.3.2 itself) is the following.

Lemma 5.5.10. *For any $i \in (s\,,\,2s]$,*

$$\Pr_{(V_i,e_i)}\Big(M_{i+1} > M_i\Big) \geqslant \frac{\alpha^2 \cdot \log^3 n}{4 \cdot \mathrm{opt}}.$$

We first identify a simple structure in the graph $\overline{G}_i$. The following claim is based on a standard low-degree orientation of the graph plus geometric grouping of degrees of vertices.

Claim 5.5.11. *<u>At least one</u> of the following two conditions is true about $\overline{G}_i$:*

(*i*) *for some $d \in \left[\alpha \cdot \log^3 n, \dfrac{\mathrm{opt}}{8\alpha}\right)$, there are $\left(\dfrac{\mathrm{opt} \cdot \log^3 n}{2d}\right)$ vertices v in Med_i with $\deg_i(v) \geqslant d$;*

(*ii*) *for some $d \in [1, \alpha \cdot \log^3 n)$, there are $\left(\dfrac{19\,\mathrm{opt} \cdot \log^3 n}{2d}\right)$ vertices in Low_i with at least d neighbors in Low_i.*

Proof. Given that High_i is empty, any edge in $\overline{G}_i$ is either incident on Med_i or is between two vertices in Low_i. Consequently, given that by $\mathcal{E}_{\mathrm{S}}(i)$, we have at least $(20\,\mathrm{opt} \cdot \log^4 n)$ edges in $\overline{G}_i$, there are either at least $(\mathrm{opt} \cdot \log^4 n)$ edges incident on Med_i or $(19 \cdot \mathrm{opt} \cdot \log^4 n)$ edges entirely inside Low_i. We prove that each case corresponds to one of the conditions in the claim.

When $\geqslant (\mathrm{opt} \cdot \log^4 n)$ edges are incident on Med_i. We partition vertices in Med_i into sets $\{D_j\}$ where each D_j contains vertices v with $\deg_i(v) \in [2^j, 2^{j+1})$. As such,

$$\sum_j |D_j| \cdot 2^{j+1} \geqslant \#\text{ edges incident on } \mathrm{Med}_i \geqslant \mathrm{opt} \cdot \log^4 n.$$

As there are at most $\log n$ choices for j in the summation above, we should have some D_{j^*} with

$$|D_{j^*}| \geqslant \frac{\mathrm{opt} \cdot \log^3 n}{2^{j^*+1}}.$$

Setting $d = 2^{j^*}$ and returning (a subset of) D_{j^*} satisfies the bound in part (*i*) of the claim: all vertices in $D_{j^*} \subseteq \mathrm{Med}_i$ have $\deg_i(\cdot)$ in $[\alpha \cdot \log^3 n, \dfrac{\mathrm{opt}}{8\alpha})$ by definition of Med_i, and we can pick a subset of D_{j^*} with size prescribed by the claim as all vertices in D_{j^*} have degree d at least.

When $\geqslant (19\,\mathrm{opt}\cdot\log^4 n)$ edges are entirely inside Low_i. The argument is almost identical to the above part by counting the degree of vertices in Low_i but only in Low_i (instead of all of $\deg_i(\cdot)$ as in the previous part). We partition vertices in Low_i into sets $\{D_j\}$ where each D_j contains all vertices with number of neighbors in Low_i in $[2^j, 2^{j+1})$. As such,

$$\sum_j |D_j|\cdot 2^{j+1} \geqslant \#\text{ edges entirely inside } \mathrm{Low}_i \geqslant 19\,\mathrm{opt}\cdot\log^4 n.$$

As there are at most $\log n$ choices for j in the summation above, we should have some D_{j^*} with

$$|D_{j^*}| \geqslant \frac{19\,\mathrm{opt}\cdot\log^3 n}{2^{j+1}}.$$

Setting $d = 2^{j^*}$ and returning (a subset of) D_{j^*} satisfies the bound in part (ii) of the claim: all vertices in $D^* \subseteq \mathrm{Low}_i$ have degree less than $(\alpha\cdot\log^3 n)$ by the definition of Low_i (even in $\overline{G}_i$ and so between Low_i also) and we can pick a subset of D_{j^*} with the required size as vertices in D_{j^*} have degree d at least. ∎ Claim 5.5.11

In the following, we refer to a step $i \in (s, 2s]$ as a ***V_i*-step** whenever case (i) of Claim 5.5.11 happens and a ***$N(V_i)$*-step** otherwise. We will show that:

- In a ***V_i*-step**, we have "enough" large degree vertices and even if we sample one of them in V_i it will make the intersection of $N(V_i)$ and $\overline{G}_i$ large;
- In a ***$N(V_i)$*-step**, we have "so many" low degree vertices in $\overline{G}_i$ that many of them will appear in $N(V_i)$ and thus there is a large intersection between $N(V_i)$ and $\overline{G}_i$ again.

In each case, we can finalize the proof by showing that having $N(V_i)$ intersect largely with $\overline{G}_i$ allows us to recover an edge e_i via $\texttt{NE-Sampler}(G, V_i)$ that can increase size of M_i with sufficiently large probability.

Case (i) of Claim 5.5.11: *V_i*-steps

Let

$$d \in [\alpha\log^3 n, \frac{\mathrm{opt}}{8\alpha}) \quad\text{and}\quad D \subseteq \mathrm{Med}_i \quad\text{with}\quad |D| = \frac{\mathrm{opt}\cdot\log^3 n}{2d} \tag{5.7}$$

be, respectively, the degree-parameter and corresponding set guaranteed by Case (i) of Claim 5.5.11. The following claim lower bounds the probability that V_i is both clean and samples a vertex from D.

Claim 5.5.12. $\Pr_{V_i}(V_i \cap D \neq \emptyset \text{ and } V_i \text{ is clean}) \geqslant \dfrac{\alpha \cdot \log^3 n}{8d}$.

Proof. We have,

$$\Pr(V_i \cap D \neq \emptyset \text{ and } V_i \text{ is clean}) \geqslant \sum_{v \in D} \Pr(v \in V_i \text{ and } V_i \text{ is clean}) - \sum_{u \neq w \in D} \Pr(u, w \in V_i)$$

(by inclusion-exclusion principle and dropping the 'intersection' from the second event)

$$\geqslant |D| \cdot \frac{3}{4k} - |D|^2 \cdot \frac{1}{k^2}$$

(by Claim 5.5.6 and as $h_i(\cdot)$ is a pair-wise independent hash function with range $[k]$)

$$= \frac{\text{opt} \cdot \log^3 n}{2d} \cdot \frac{\alpha}{\text{opt}} \cdot \left(\frac{3}{4} - \frac{\text{opt} \cdot \log^3 n \cdot \alpha}{2d \cdot \text{opt}}\right)$$

(by the choice of $k = \text{opt}/\alpha$ and size of D in Eq (5.7))

$$\geqslant \frac{\alpha \cdot \log^3 n}{8d},$$

as $d \geqslant \alpha \log^3 n$ by Eq (5.7). ∎ Claim 5.5.12

Let us now condition on the choice of V_i and assume the event of Claim 5.5.12 happens. Given that any vertex in D already has d neighbors in $\overline{G}_i$, we have that $N(V_i) \cap \overline{G}_i$ has size at least d in this case. On the other hand, $N(V_i)$ can have at most $(\text{opt}/4\alpha)$ neighbors outside $\overline{G}_i$ by the bound on the total number of matched vertices by $\mathcal{E}_{\text{M}}(i)$. As the choice of e_i from `NE-Sampler`(G, V_i) is uniform over $N(V_i)$, we have,

$$\begin{aligned}
\Pr_{e_i}\big(e_i \text{ is from } V_i \text{ to } N(V_i) \cap \overline{G}_i \mid V_i\big) &\geqslant (1 - \delta_F) \cdot \frac{\big|N(V_i) \cap \overline{G}_i\big|}{|N(V_i)|} \\
&\geqslant (1 - \delta_F) \cdot \frac{d}{(\text{opt}/4\alpha) + d} \\
&\geqslant (1 - \delta_F) \cdot \frac{d \cdot 8\alpha}{3\text{opt}} \quad (\text{as } d \leqslant (\text{opt}/8\alpha) \text{ in Eq (5.7)}) \\
&\geqslant \frac{2d \cdot \alpha}{\text{opt}},
\end{aligned}$$

as $\delta_F < 1/4$. Given that all of V_i is also unmatched (as V_i is clean by conditioning on the event of Claim 5.5.12), we can include e_i in M_{i+1} greedily whenever e_i is between V_i and $N(V_i) \cap \overline{G}_i$.

Consequently, combining the two events above, we have,

$$\Pr_{(V_i,e_i)}\Big(M_{i+1} > M_i\Big) \geqslant \Pr_{V_i}(V_i \cap D \neq \emptyset \text{ and } V_i \text{ is clean}) \cdot \Pr_{e_i}\big(e_i \text{ is from } V_i \text{ to } N(V_i) \cap \overline{G}_i \mid V_i\big)$$
$$\geqslant \frac{\alpha \log^3 n}{8d} \cdot \frac{2d \cdot \alpha}{\text{opt}} = \frac{\alpha^2 \log^3 n}{4\,\text{opt}}.$$

This concludes the proof of Lemma 5.5.10 in this case.

Case (ii) **of Claim 5.5.11:** $\boldsymbol{N(V_i)}$**-steps**

Let

$$d \in [1, \alpha \log^3 n) \quad \text{and} \quad D \subseteq \text{Low}_i \quad \text{with} \quad |D| = \frac{19\,\text{opt} \cdot \log^3 n}{2d} \tag{5.8}$$

be, respectively, the degree-parameter and corresponding set guaranteed by Case (ii) of Claim 5.5.11. For the rest of this analysis, we focus only on the subgraph of $\overline{G}_i$ induced on vertices of Low_i and for each $v \in D$, we pick exactly d (arbitrary) neighbors from Low_i and denote them by $NL(v)$. Our goal is to show that $N(V_i)$ and $\overline{G}_i$ intersect largely. We will do so by counting the elements in D that have neighbors in V_i. This works because $D \subseteq \overline{G}_i$ and having neighbors in V_i means that the vertex itself is in $N(V_i)$.

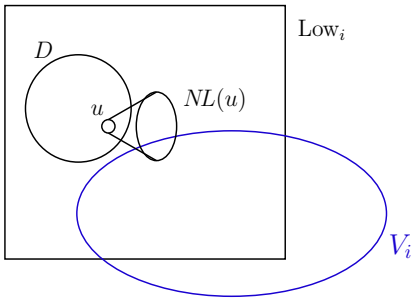
For any vertex $v \in D$, define an indicator random variable $X_v \in \{0, 1\}$ which is 1 iff $NL(v) \cap V_i \neq \emptyset$ (see Figure 5.5a). Notice that $X = \sum_{v \in D} X_v$ is a random variable that denotes the number of vertices v in D that have a neighbor in $NL(v)$ that belongs to V_i. Note that we do not consider all neighbors of v in Low_i, only the ones in $NL(v)$; this is okay since we just need a lower bound on $\left|N(V_i) \cap \overline{G}_i\right|$. It is easy to see that $X \leqslant \left|N(V_i) \cap \overline{G}_i\right|$ since v contributes to $\left|N(V_i) \cap \overline{G}_i\right|$ if $X_v = 1$ (see Figure 5.5b). We first bound the probability of the event $NL(v) \cap V_i \neq \emptyset$.

Claim 5.5.13. *For any $v \in D$,*

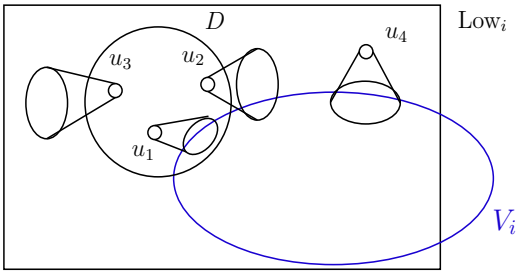
$$(1 - o(1)) \cdot \frac{d \cdot \alpha}{\text{opt}} \leqslant \Pr(X_v = 1) \leqslant \frac{d \cdot \alpha}{\text{opt}}.$$

Proof. $X_v = 1$ iff one of the neighbors of v in $NL(v)$ belongs to V_i. For the upper bound, by union bound,

$$\Pr(X_v = 1) \leqslant \sum_{u \in NL(v)} \Pr(u \in V_i) = |NL(v)| \cdot \frac{1}{k} = \frac{d \cdot \alpha}{\text{opt}}. \quad (\text{as } |NL(v)| = d \text{ and } k = \text{opt}/\alpha)$$



(a) This figure shows the set of vertices Low_i and its subset D. V_i (in blue) is the set of sampled vertices in step i. For a vertex u in D we define a set of neighbors $NL(u)$ which if intersects with V_i then we have random variable $X_u = 1$.



(b) This figure shows $X \leqslant \left|N(V_i) \cap \overline{G}_i\right|$. The vertices u_j are in Low_i. Notice that $X_{u_1} = 1, X_{u_2} = 1, X_{u_3} = 0$ and X_{u_4} is not defined so we have $X = 2$. But $\left|N(V_i) \cap \overline{G}_i\right| = 3$ since u_1, u_2 and u_4 contribute to it.

Figure 5.5: Illustration of random variables X_v.

For the lower bound, by inclusion-exclusion principle,

$$\begin{aligned}
\Pr(X_v = 1) &\geqslant \sum_{u \in NL(v)} \Pr(u \in V_i) - \sum_{u \neq w \in NL(v)} \Pr(u, w \in V_i) \\
&\geqslant |NL(v)| \cdot \frac{1}{k} - |NL(v)|^2 \cdot \frac{1}{k^2} \\
&= \frac{d \cdot \alpha}{\text{opt}} \cdot \left(1 - \frac{d \cdot \alpha}{\text{opt}}\right) && (\text{as } |NL(v)| = d \text{ and } k = \text{opt}/\alpha) \\
&\geqslant (1 - o(1)) \cdot \frac{d \cdot \alpha}{\text{opt}},
\end{aligned}$$

as $d < \alpha \log^3 n$ by Eq (5.8) and $\text{opt} \geqslant \alpha^2 \cdot n^\delta$ by Assumption 2. ∎ Claim 5.5.13

By Claim 5.5.13 and the size of D in Eq (5.8), we have,

$$(1 - o(1)) \cdot \frac{19}{2} \cdot \alpha \cdot \log^3 n \leqslant \mathbb{E}[X] \leqslant \frac{19}{2} \cdot \alpha \cdot \log^3 n. \tag{5.9}$$

Our goal now is to prove that X is concentrated. This requires a non-trivial proof as the variables $\{X_v\}_{v \in D}$ are correlated through their shared neighbors in V_i. But the fact that the subgraph induced on Low_i is low-degree allows us to bound the variance of X using a combinatorial argument in the following claim.

Claim 5.5.14. $\text{Var}[X] \leqslant (1/8) \cdot \mathbb{E}[X]^2$.

Proof. For any two vertices $u \neq v \in D$, define $\mathrm{Com}(u,v) := NL(u) \cap NL(v)$ as the set of common neighbors of u and v in subgraph of Low_i defined by $NL(\cdot)$ and let $\mathrm{com}(u,v) = |\mathrm{Com}(u,v)|$. We have,

$$\mathrm{Var}[X] = \sum_{v \in D} \mathrm{Var}[X_v] + \sum_{u \neq v \in D} \mathrm{Cov}[X_u, X_v] \leqslant \mathbb{E}[X] + \sum_{u \neq v \in D} \mathrm{Cov}[X_u, X_v], \tag{5.10}$$

as X_v is an indicator random variable and thus $\mathrm{Var}[X_v] \leqslant \mathbb{E}[X_v]$. We thus need to bound the covariance-terms only. Recall that

$$\begin{aligned}\mathrm{Cov}[X_u, X_v] &= \mathbb{E}[X_u \cdot X_v] - \mathbb{E}[X_u]\,\mathbb{E}[X_v] \\ &= \Pr(NL(u) \cap V_i \neq \emptyset \wedge NL(v) \cap V_i \neq \emptyset) - \Pr(X_u = 1) \cdot \Pr(X_v = 1).\end{aligned}$$

We can bound the second part using Claim 5.5.13 for each probability-term. For the first part, notice that for $NL(u) \cap V_i \neq \emptyset$ and $NL(v) \cap V_i \neq \emptyset$ one of the following two things should happen: at least one of the shared neighbors of u, v in $\mathrm{Com}(u,v)$ is chosen in V_i or each of them separately have a neighbor in $NL(u) - \mathrm{Com}(u,v)$ and $NL(v) - \mathrm{Com}(u,v)$ those join V_i (as $h_i(\cdot)$ is a pair-wise independent hash function, the probability of these two distinct vertices joining V_i is independent). Thus,

$$\begin{aligned}\Pr(NL(u) \cap V_i \neq \emptyset \wedge NL(v) \cap V_i \neq \emptyset) &\leqslant \sum_{w \in \mathrm{Com}(u,v)} \Pr(w \in V_i) + \sum_{\substack{z_u \in NL(u) - \mathrm{Com}(u,v) \\ z_v \in NL(v) - \mathrm{Com}(u,v)}} \Pr(z_u \in V_i) \cdot \Pr(z_v \in V_i) \\ &\leqslant \mathrm{com}(u,v) \cdot \frac{1}{k} + d^2 \cdot \frac{1}{k^2}\end{aligned}$$

(as $h_i(\cdot)$ is uniform over $[k]$ and $u, v \in D$ and each vertex in D has exactly d neighbors in $NL(\cdot)$)

$$= \mathrm{com}(u,v) \cdot \frac{\alpha}{\mathrm{opt}} + \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2}. \qquad (\text{as } k = \mathrm{opt}/\alpha)$$

Plugging in this for the first term of covariance and the bounds in Claim 5.5.13 for the second terms, we have,

$$\mathrm{Cov}[X_u, X_v] \leqslant \mathrm{com}(u,v) \cdot \frac{\alpha}{\mathrm{opt}} + \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2} - (1 - o(1)) \cdot \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2} = \mathrm{com}(u,v) \cdot \frac{\alpha}{\mathrm{opt}} + o(1) \cdot \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2}.$$

By plugging in further in the RHS of Eq (5.10), we get that,

$$\mathrm{Var}[X] \leqslant \mathbb{E}[X] + \left|D^2\right| \cdot o(1) \cdot \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2} + \sum_{u \neq v \in D} \mathrm{com}(u,v) \cdot \frac{\alpha}{\mathrm{opt}}$$

$$\leqslant \mathbb{E}[X] + \left(\frac{19 \cdot \mathrm{opt} \cdot \log^3 n}{2d}\right)^2 \cdot o(1) \cdot \frac{d^2 \cdot \alpha^2}{\mathrm{opt}^2} + \sum_{u \neq v \in D} \mathrm{com}(u,v) \cdot \frac{\alpha}{\mathrm{opt}}$$

(by the bound on size of D in Eq (5.8))

$$\leqslant o(1) \cdot \mathbb{E}[X]^2 + \frac{\alpha}{\mathrm{opt}} \cdot \sum_{u \neq v \in D} \mathrm{com}(u,v).$$

(by the lower bound on $\mathbb{E}[X] \geqslant (1 - o(1)) \cdot (19/2) \cdot \alpha \cdot \log^3 n$ in Eq (5.9))

The remaining part is then to compute the summation in the RHS which we do below using a double-counting argument. Note that $\sum_{u \neq v \in D} \mathrm{com}(u,v)$ counts the number of common neighbors inside $NL(\cdot)$-subgraph of Low_i for each pair of vertices in D. This can be alternatively counted by going over vertices in Low_i that are neighbor to D and count the number of pairs of neighbors (in $NL(\cdot)$) they have in D.

$$\sum_{u \neq v \in D} \mathrm{com}(u,v) = \sum_{z \in NL(D)} \binom{|NL(z)|}{2} \leqslant \sum_{z \in NL(D)} |NL(z)|^2$$

$$\leqslant (\alpha \cdot \log^3 n) \cdot \sum_{z \in NL(D)} |NL(z)|$$

(each vertex in Low_i has degree at most $(\alpha \cdot \log^3 n)$ to Low_i and $z \in \mathrm{Low}_i$ as it is in $NL(D)$)

$$\leqslant (\alpha \cdot \log^3 n) \cdot |D| \cdot d$$

(as the sum-term counts the number of edges between D and $NL(D)$ which is at most $|D| \cdot d$)

$$\leqslant (\alpha \cdot \log^3 n) \cdot (\frac{19}{2} \cdot \mathrm{opt} \cdot \log^3 n).$$ (by Eq (5.8) on the size of D)

Plugging in this bound in the upper bound on $\mathrm{Var}[X]$ in the earlier equation, we have,

$$\begin{aligned}
\mathrm{Var}[X] &\leqslant o(1) \cdot \mathbb{E}[X]^2 + \frac{\alpha}{\mathrm{opt}} \cdot \left(\frac{19}{2} \cdot \mathrm{opt} \cdot \alpha \cdot \log^6 n\right) \\
&= o(1) \cdot \mathbb{E}[X]^2 + \frac{19}{2} \cdot \alpha^2 \cdot \log^6 n \\
&< \frac{1}{8} \cdot \mathbb{E}[X]^2,
\end{aligned}$$

as $\mathbb{E}[X] \geqslant (1 - o(1)) \cdot \frac{19}{2} \cdot (\alpha \cdot \log^3 n)$ by Eq (5.9). ∎ Claim 5.5.14

Recall that

$$\left|N(V_i) \cap \overline{G}_i\right| \geqslant X.$$

Given the bound on expectation and variance of X in Eq (5.9) and Claim 5.5.14, respectively, we can now apply Chebyshev's inequality and get that,

$$\Pr\left(\left|N(V_i)\cap\overline{G}_i\right| < 2\alpha\cdot\log^3 n\right) \leqslant \Pr\left(\left|X-\mathbb{E}\left[X\right]\right| \geqslant \frac{15}{19}\cdot\mathbb{E}\left[X\right]\right) \leqslant \frac{19^2\cdot\mathrm{Var}\left[X\right]}{15^2\cdot\mathbb{E}\left[X\right]^2} \leqslant \frac{19^2}{15^2\cdot 8} < \frac{1}{4}.$$

Additionally, we also have that the probability that V_i is not clean is at most,

$$\Pr(V_i \text{ is not clean}) \leqslant \sum_{u\in V(M_i)} \Pr(u\in V_i) < \frac{\mathrm{opt}}{4\alpha}\cdot\frac{\alpha}{\mathrm{opt}} = \frac{1}{4}.$$

By a union bound on the two equations above, we have,

$$\Pr\left(\left|N(V_i)\cap\overline{G}_i\right| \geqslant 2\alpha\cdot\log^3 n \text{ and } V_i \text{ is clean}\right) \geqslant \frac{1}{2}. \tag{5.11}$$

The rest of the proof is similar to that of ***V_i*-steps**. We condition the choice of V_i and assume the event of Eq (5.11) has happened. Thus, we have that both V_i is clean and $N(V_i)$ has at least $2\alpha\cdot\log^3 n$ vertices in $\overline{G}_i$. Moreover, $N(V_i)$ can have at most $(\mathrm{opt}/4\alpha)$ neighbors outside $\overline{G}_i$ by the bound on the total number of matched vertices by $\mathcal{E}_{\mathrm{M}}(i)$. As the choice of e_i from `NE-Sampler`(G, V_i) is uniform over $N(V_i)$, we have,

$$\begin{aligned}
\Pr_{e_i}\left(e_i \text{ is from } V_i \text{ to } N(V_i)\cap\overline{G}_i \mid V_i\right) &\geqslant (1-\delta_F)\cdot\frac{\left|N(V_i)\cap\overline{G}_i\right|}{|N(V_i)|}\\
&\geqslant (1-\delta_F)\cdot\frac{2\alpha\cdot\log^3 n}{(\mathrm{opt}/4\alpha)+2\alpha\cdot\log^3 n}\\
&> \frac{3\cdot\alpha^2\cdot\log^3 n}{\mathrm{opt}},
\end{aligned}$$

as $\mathrm{opt}\geqslant\alpha^2\cdot n^{\delta}$ by Assumption 1 and $\delta_F < 1/2$. Given that all of V_i is also unmatched (as V_i is clean), we can include e_i in M_{i+1} greedily whenever the event of the LHS above happens.

Consequently, combining the two events above, we have,

$$\begin{aligned}
\Pr_{(V_i,e_i)}\left(M_{i+1} > M_i\right) &\geqslant \Pr_{V_i}\left(\left|N(V_i)\cap\overline{G}_i\right| \geqslant 2\alpha\cdot\log^3 n \text{ and } V_i \text{ is clean}\right)\cdot\Pr_{e_i}\left(e_i \text{ is from } V_i \text{ to } N(V_i)\cap\overline{G}_i \mid V_i\right)\\
&\geqslant \frac{1}{2}\cdot\frac{3\cdot\alpha^2\cdot\log^3 n}{\mathrm{opt}} > \frac{\alpha^2\log^3 n}{\mathrm{opt}}.
\end{aligned}$$

This concludes the proof of Lemma 5.5.10 in this case also.

Concluding the Proof of Lemma 5.5.4

By Lemma 5.5.10, assuming the events $\mathcal{E}_{\mathrm{M}}(i), \mathcal{E}_{\mathrm{S}}(i)$ hold for every $i \in (s, 2s]$, size of M_{2s} statistically dominates sum of s independent Bernoulli random variables $\{Z_i\}_{i=1}^{s}$ with mean $(\alpha^2 \cdot \log^3 n/4\mathrm{opt})$ (RHS of Lemma 5.5.10). Let $Z = \sum_{i=1}^{s} Z_i$. Thus, by the choice of s in Algorithm 11,

$$\mathbb{E}[Z] = \frac{\mathrm{opt}^2}{(\alpha \cdot \log n)^3} \cdot \frac{\alpha^2 \cdot \log^3 n}{4 \cdot \mathrm{opt}} = \frac{\mathrm{opt}}{4 \cdot \alpha},$$

and by the Chernoff bound (Proposition 2.4.3),

$$\Pr\left(Z < \frac{\mathrm{opt}}{8\alpha}\right) < \Pr\left(Z < \frac{1}{2} \cdot \mathbb{E}[Z]\right) \leqslant \exp\left(-\frac{\mathrm{opt}}{12\alpha}\right) \ll 1/\mathrm{poly}(n),$$

where the final bound is by Assumption 2 as $\mathrm{opt} \geqslant n^{\delta} \cdot \alpha^2$. This means that as long as $\mathcal{E}_{\mathrm{M}}(i), \mathcal{E}_{\mathrm{S}}(i)$ happen for all $i \in (s : 2s]$, with high probability we are going to end up with a matching M_{easy} of size at least $(\mathrm{opt}/8\alpha)$, which means $\mathcal{E}_{\mathrm{M}}(2s)$ does not happen as desired.

This concludes the proof of Lemma 5.3.2, and combined with Lemma 5.3.3, the entire proof of Theorem 9.

Chapter 6

Estimating Correlation Clustering Cost in Streaming

6.1 Introduction

Correlation clustering is a fundamental optimization problem at the intersection of machine learning and theoretical computer science. This problem was introduced by the work of [BBC04][1], with motivation to document clustering, as follows: we have a complete graph $G = (V, E)$ whose edges are labeled by either $(+)$ or $(-)$, and the objective, known as *disagreement minimization*, is to cluster the vertices so that the number of $(+)$ edges across clusters and $(-)$ edges inside the same clusters are minimized. Correlation clustering has since found broad applications in areas such as document categorization [BBC04], webpage segmentation [CKP08], microscopy imaging [ZYH14], and community detection [VGW18, SDE+21], to name a few.

There is an abundant body of literature studying polynomial time algorithms for correlation clustering. [BBC04, CMSY15] showed that there exists a 2.06-approximation algorithm in polynomial time and the problem is NP-hard and even APX-hard [CGW03] with constant 24/23 [CCAL+24]. Furthermore, [ACN05] gave a simple (combinatorial) poly-time 3-approximation algorithm that is widely used in practice. Very recently, breakthrough results by [CLN22, CALLN23] achieved 1.73 approximation in polynomial time. Most recently, [CCAL+24, CCAL+25b] obtain a $(1.437 + \varepsilon)$-approximation in polynomial time which is state of the art. In addition, efficient algorithms are also explored for several variants of this problem like the agreement maximization objective [BBC04, Swa04,

[1] Historically, some forms of the problem were mentioned in earlier works like [GW89].

CGW03], weighted graphs [Swa04, CGW03, EF03], fixed number of clusters [GG06], fair clustering [AEKM20, SZ22], and others.

In recent years, with the rapid development of the 'big data era', there has been a growing interest in algorithms for correlation clustering under *sublinear* models. In general, learning algorithms under sublinear models are able to output the answer without processing or storing the *entire input*. For instance, [BGK13, AW22] gave algorithms to (approximately) learn the clustering in sublinear time, [CLM+21, BCMT22] designed algorithms in the Massively Parallel Computation (MPC) model, and [BDP+25, CALMP24, CCAL+25a] designed algorithms in the dynamic model with $\text{polylog}\, n$ update time. Another widely popular model of sublinear algorithms—the focus of our paper—is the *graph streaming model*. In this model, the edges of the input graph are given to the algorithm one by one in a stream and the memory of the algorithm is desired to be substantially smaller than the input. Here, [ACG+15] designed a streaming algorithm with $\widetilde{O}(n)$ memory that achieves 3-approximation with $O(\log\log(n))$ passes – here, n is the number of vertices of the graph and thus the input size is $\Theta(n^2)$ bits. The number of passes of these algorithms were later improved to $O(1)$ by [CLM+21] and a single pass by [AW22], albeit with much larger, yet still a constant, approximation factor. Very recently, [BCMT23] further improved the approximation ratio of single-pass streaming algorithms to a 5-approximation in polynomial time and $(1+\varepsilon)$-approximation in exponential-time. [MC23, CKL+24] improved this to a 3-approximation in polynomial time that also works in dynamic streams where edge insertions and deletions are allowed. Finally, [AKP25] almost resolved the correlation clustering problem in semi-streaming space and polynomial time by giving an $(\alpha + o(1))$-approximation where α is the best approximation ratio of any polynomial time classical algorithm for correlation clustering.

The aforementioned line of works focused on the $\widetilde{O}(n)$-memory regime. Allowing for $\Omega(n)$ memory in these algorithms is necessary given that even outputting the solution, namely, the clustering of the input labeled graph, requires this much memory. Yet, in many application, $\Omega(n)$ memory can still be quite large, and the implementations can require significant resource. In such cases, it is highly desirable to determine the "clusterability" of the input graph *before* running the actual clustering algorithm. If even the optimal cost for the input is high, it implies the clustering cannot provide any meaningful outcome, and we should not waste resources on these instances. This type of "value estimation" problem is extensively studied in the streaming literature, see e.g. [KKS15, GT19, ACL+22, BOS22, DKPP22] for several examples in this context. This raises the following fundamental question:

How well can we ***estimate*** *the optimal correlation clustering* ***cost*** *with* $\text{polylog}\,(n)$-

space streaming algorithms?

Despite the vast body of work on streaming correlation clustering in general, this question has received almost no attention so far. Indeed, to the best of our knowledge, the only prior work here is that of [ACG+15] who proved that any (finite) *purely multiplicative* factor approximation of the cost is not possible in $o(n)$ space. This result however does *not* rule out any *additive error* approximation (say, in the spirit of [BGK13] for local query algorithms).

We remedy this state of affairs in this paper. Our main algorithmic results show that one can obtain a 1-approximation plus $o(n^2)$ additive approximation to the cost of optimal correlation clustering in only $\text{polylog}\,(n)$ space. We also give a more practical algorithm with a 3-approximation plus $o(n^2)$ additive approximation in only $\text{polylog}\,(n)$ space – the additive approximation can be further reduced at the cost of increasing the multiplicative approximation to some large constant. We then complement these results with new streaming lower bounds that further justify the necessity of the additive errors in our algorithms. Throughout, we will present algorithms in the manner of insertion-only streams. However, we shall note that our algorithms can be extended to *dynamic* streams wherein the edges of the graph can be both inserted and deleted during the stream.

6.1.1 Our Contributions

To state our results, we need the following notation. Let OPT be the optimal cost for correlation clustering. We say an algorithm achieves an (α, β)-approximation to OPT if it gives an α-multiplicative with a β additive approximation, namely, outputs a number $\textsc{alg}$ such that

$$\mathsf{OPT} \leqslant \textsc{alg} \leqslant \alpha \cdot \mathsf{OPT} + \beta.$$

Our main result is a single-pass streaming algorithm that achieves a $(1, \delta n^2)$-approximation with high probability in $\text{poly}(\log n, 1/\delta)$ space.

Result 6.1. *There is a single-pass dynamic streaming algorithm that with high probability gives a $(1, \delta n^2)$-approximation for the correlation clustering value and takes $O(\text{polylog}\, n/\text{poly}(\delta))$ words of space.*

This algorithm achieves a purely additive approximation but has a post-processing time of $O(\log^2 n \cdot (1/\delta)^{O(1/\delta^2 \cdot \log^2 1/\delta)})$. This exponential dependence on $1/\delta$ in the runtime is practically undesirable, so we also propose additional algorithms with significantly improved post-processing times. They leverage sparse-dense decomposition and pivot-based

techniques, offering faster and more practical algorithms for estimating the correlation clustering cost despite their weaker theoretical guarantees on the approximation factor.

The first one is a single-pass streaming algorithm that achieves an $(O(1), \delta n^2)$-approximation with high probability in $\operatorname{polylog} n/\operatorname{poly}(\delta)$ space.

Result 6.2. *There is a single-pass dynamic streaming algorithm that outputs an $(O(1), \delta n^2)$-approximation of the optimal correlation clustering cost with high probability and uses space $O(\operatorname{polylog}(n)/\delta^5)$ words.*

This algorithm offers an improved post-processing time of $\operatorname{polylog} n/\operatorname{poly}(\delta)$ and exhibits a better dependence on δ compared to Result 6.1. However, this improvement comes at the cost of a large constant multiplicative factor. The algorithm is based on the sparse-dense decomposition framework from [AW22], which inherently introduces a worst-case constant of approximately 10^7. Our next result addresses this drawback: we present an alternative algorithm that achieves a $(3, \delta n^2)$-approximation in expectation with $\operatorname{polylog} n$ space at the cost of an exponential dependence on δ in the space.

Result 6.3. *There is a single-pass dynamic streaming algorithm that in expectation outputs a $(3, \delta n^2)$-approximation of the optimal correlation clustering cost and uses $2^{O(1/\delta)} \cdot \operatorname{polylog}(n)$ space.*

This algorithm has an exponential update time in $1/\delta$, but its post-processing time is only $\operatorname{polylog} n/\operatorname{poly}(\delta)$. Compared to the algorithm in Result 6.2, the algorithm in Result 6.3 is better suited for instances where the optimal correlation clustering cost is large (e.g., $\mathsf{OPT} = \Omega(n^2)$). In such cases, we can set δ to be a small constant and still achieve a $(3+\delta)$-multiplicative approximation using only $\operatorname{polylog}(n)$ space.

Since all of our upper bounds contain the δn^2 additive error, it is natural to ask *to what extent these additive terms are necessary*. We present two new lower bounds that partially address this question and show that such additive errors are, to a large extent, unavoidable. Our first lower bound demonstrates that if *only* additive error is allowed, then no single-pass streaming algorithm using $\operatorname{polylog}(n)$ space can achieve an additive error substantially better than $O(n^2)$.

Result 6.4. *No single-pass streaming algorithm with $\operatorname{polylog}(n)$ space can output a $(1, n^{2-\varepsilon})$ approximation of the optimal correlation clustering cost with a sufficiently large constant probability of success for some $\varepsilon = o(1)$.*

This result shows that our algorithm in Result 6.1 is almost optimal for constant δ. Moreover, Result 6.4 can be viewed as the *additive* analogue of the lower bounds presented in [ACG+15], which focuses instead on a purely multiplicative approximation.

We have almost resolved the complexity of the problem in the regime where a $\Theta(n^2)$ additive error is allowed. A natural next question is whether permitting a small multiplicative approximation might eliminate the need for any additive error altogether. To address this, we present a second lower bound showing that even when a small multiplicative approximation is allowed, an additive error of at least $\Theta(n)$ remains necessary.

Result 6.5. *No single-pass streaming algorithm with even $o(\sqrt{n})$ space can output a $(1.19, O(n))$ approximation of the optimal correlation clustering cost with a sufficiently large constant probability of success.*

We now discuss the scenarios where our algorithms are most effective. To distinguish between instances with low versus high correlation clustering costs (e.g., $o(n^2)$ vs. $\Omega(n^2)$), it suffices to run the algorithm from Result 6.2 with $\delta = 1/\log(n)$, which requires only $\widetilde{O}(1)$ space. To separate instances that are *partially clusterable* (e.g., with an optimal cost of $n^2/1000$) from those that are *not clusterable at all* (e.g., optimal cost of $n^2/5$), the algorithm from Result 6.3, run with a small constant $\delta = O(1)$, also achieves this using $\widetilde{O}(1)$ space. The only regime our algorithms are not well-suited for is distinguishing between multiple *well-clusterable* instances, a setting that appears to have limited practical relevance.

Among our algorithms, Result 6.1 offers the best of both worlds—achieving stronger approximation guarantees and polynomial dependence on $1/\delta$. As long as post-processing time is not a primary concern, this algorithm is the preferred choice. In the remainder of this chapter, we will focus exclusively on this upper bound, along with the two accompanying lower bounds.

6.1.2 Our Techniques

Upper bound of Result 6.1. Our upper bound is derived by reducing the correlation clustering value problem—with $1/\delta$ clusters and δn^2 additive error—to an instance of Max CSP. We begin by observing that if we restrict the solution to use at most $1/\delta$ clusters, rather than allowing an arbitrary number, the optimal cost increases by at most $O(\delta n^2)$. To prove this, we take the optimal clustering and merge smaller clusters to form clusters of size $\Theta(\delta n)$. This merging process increases the cost by at most $\Theta(\delta n)$ per vertex, leading to a total additive error of at most $\Theta(\delta n^2)$. Importantly, this transformation ensures the number of clusters is bounded by $O(1/\delta)$.

Since we are considering only additive error, we may equivalently focus on maximizing the agreement cost rather than minimizing the disagreement cost. The problem of maximizing agreement in correlation clustering with at most $1/\delta$ clusters can be formulated as an instance of Max 2-CSP. A result by [BHHS11] shows that the optimal agreement value with $1/\delta$ clusters can be approximated in expectation up to an additive error of δn^2 by sampling a uniform subset of $\text{poly}(1/\delta)$ vertices, solving the problem on the induced subgraph, and appropriately rescaling the cost.

Based on this, our streaming algorithm samples a small subgraph of size $k := \text{poly}(1/\delta)$ and solves the CSP instance on this induced subgraph with an additive error of δk^2 after the stream. To get success with high probability, we repeat this process in parallel and take the median result. Since each step introduces an additive error of at most $O(\delta n^2)$, the final rescaled output incurs an overall error of at most δn^2.

Lower bounds. Our lower bounds are mostly adapted from well-known communication problems, albeit with some novel tricks. In particular, our lower bound in Result 6.4 borrows idea from a recent reduction from INDEX in [AAD+23], which was used to give an $\Omega(n^2)$ space lower bound to return the *exact* correlation clustering cost in a single pass. We prove that the same space-error trade-off holds even if we duplicate each vertex multiple times. As such, we can obtain a lower bound of $\Omega(n^\varepsilon)$ for any algorithm with only $O(n^{2-\varepsilon})$ additive error. Our lower bound in Result 6.5 follows from a reduction from the famous Gap-Cycle Counting (GCC) problem. En route to proving the lower bound, we slightly tweak the original instance of Gap-cycle Counting, and show that the lower bound works also for *odd* cycles, which turns out to be necessary to separate the correlation clustering costs.

6.2 Preliminaries

We introduce the necessary notation, the problem definition, and the standard technical tools we are going to use in this section.

6.2.1 Notation

We use $G = (V, E^+ \cup E^-)$ to denote a labeled complete graph arriving in a stream. For any vertex v, we denote by $N^+(v)$ all vertices that have an $(+)$ edge from v, and we let $N^+[v] = \{v\} \cup N^+(v)$. Similarly, we denote by $N^-(v)$ all vertices in the vertex set V that have an $(-)$ edge from v, and we let $N^-[v] = \{v\} \cup N^-(v)$. We use $N^+(u) \triangle N^+(v)$ (resp. $N^-(u) \triangle N^-(v)$) to denote the disjoint neighborhoods of u and v, i.e.

$$N^+(u) \triangle N^+(v) = (N^+(u) \cup N^+(v)) \setminus (N^+(u) \cap N^+(v)).$$

For a fixed set of vertices $A \subseteq V$, we further let $E^+(v, A)$ be the set of $(+)$ edges between v and vertices in A, and $E^-(v, A)$ be the set of $(-)$ edges between v and vertices in A.

We use $G^+ = (V, E^+)$ to denote the positive subgraph of G with all the $(+)$ edges, and we define $G^- = (V, E^-)$ analogously. Note that since we work with a labeled complete graph, the information of G^- edges can be uniquely inferred from the positive subgraph G^+. As such, when we work with G^+ only, we call a $(+)$ edge $(u, v) \in E^+$ as an *edge in* G^+. Similarly, we call a $(-)$ edge $(u, v) \in E^-$ an *non-edge in* G^+. We may omit G^+ when the context is clear.

For a fixed cluster $\mathcal{C}$ on a labeled complete graph G, we use $\mathrm{CC}(\mathcal{C})$ to denote the total cost of correlation clustering on G by $\mathcal{C}$ (see Problem 6 for the formal definition of cost). Furthermore, we slightly abuse the notation to reload $\mathrm{CC}(\cdot)$ as a function of cost on subgraphs of G in the following occasions:

1. For an induced subgraph $H \subseteq G$, we let $\mathrm{CC}(\mathcal{C}, H)$ be the cost of $\mathcal{C}$ induced by the edges with both endpoints in H;
2. For two vertex sets $A, B \subseteq V$, we let $\mathrm{CC}(\mathcal{C}, (A, B))$ be the cost of $\mathcal{C}$ induced by the edges with exactly one endpoint in A and one endpoint in B.

Finally, for a single edge (u, v), we use the notation $\mathsf{edge\text{-}cost}(\mathcal{C}, (u, v))$ to denote the cost induced by a single edge (u, v) for clustering $\mathcal{C}$. We may write a short-hand notation $\mathsf{edge\text{-}cost}(\,(u, v))$ when the context is clear that $\mathcal{C}$ is used.

6.2.2 Problem Definition

We now give the formal description of the problem.

Problem 6 (Correlation Clustering Value Estimation)**.** *Given a labeled complete graph $G = (V, E^- \cup E^+)$ and a clustering $\mathcal{C}$ that partitions V into disjoint set of vertices $C_1, C_2, \cdots, C_k$, the cost of disagreement minimization correlation clustering on $\mathcal{C}$ is defined as*

$$CC(\mathcal{C}) := \left|\left\{(u, v) \in E^+ \mid \exists i \neq j \text{ s.t. } u \in C_i, v \in C_j\right\}\right| + \left|\left\{(u, v) \in E^- \mid \exists i \text{ s.t. } u, v \in C_i\right\}\right|$$

Let OPT *be the minimum CC over all possible valid clusterings. The Correlation Clustering Value Estimation problem asks for a number* ALG *such that* $\mathsf{OPT} \leqslant \textsc{alg} \leqslant f(\mathsf{OPT})$ *for some function* $f(x) \geqslant x$.

If $f(\mathsf{OPT}) = \alpha \cdot \mathsf{OPT} + \beta \cdot n^2$ for α and β, we say that ALG is a (α, β)-approximation of the value of correlation clustering in this scenario.

6.2.3 The Streaming Model

We study algorithms for correlation clustering value estimation under the *graph streaming* model, where edges arrive one after another in a stream with the labels. We introduce the graph streaming model for the labeled complete graph for both insertion-only and dynamic streams.

Insertion-only labeled complete graph stream. Given a graph $G = (V, E^+ \cup E^-)$, the insertion-only graph stream of G is a length-T sequence of tuples $\langle \sigma_1, \sigma_2, \cdots, \sigma_T \rangle$ such that $T = \binom{n}{2}$, and each $\{\sigma_t\}_{t=1}^T$ is consistent of

$$\sigma_t = ((u_t, v_t), x_t),$$

where (u_t, v_t) specifies a unique vertex pair and x_t gives the label of (u_t, v_t).

Dynamic labeled complete graph stream. Given a graph $G = (V, E^+ \cup E^-)$, the dynamic graph stream of G is a length-T sequence of tuples $\langle \sigma_1, \sigma_2, \cdots, \sigma_T \rangle$ such that $T \geqslant \binom{n}{2}$, and each $\{\sigma_t\}_{t=1}^T$ is consistent of

$$\sigma_t = ((u_t, v_t), \Delta_t),$$

where (u_t, v_t) specifies a vertex pair and Δ_t gives the *update* of the label on (u_t, v_t) with the following rules:

- If (u_t, v_t) does *not* have any $(+)$ or $(-)$ edges, Δ_t can be the insertion of $(+)$ or $(-)$ labels, but cannot be a deletion operation.
- If (u_t, v_t) has at least one $(+)$ label, Δ_t can be the insertion or deletion of a $(+)$ edge, but it cannot specify any operation for $(-)$ edges.
- If (u_t, v_t) has at least one $(-)$ label, Δ_t can be the insertion or deletion of a $(-)$ edge, but it cannot specify any operation for $(+)$ edges.

In other words, the dynamic stream is allowed to perform insertions and deletions whenever it is *consistent* with the existing labels on the vertex pair. Finally, the stream has to provide a labeled complete graph in the end, i.e. every pair of vertices has exactly one $(+)$ or $(-)$ edge.

We note that our algorithms work even when given a dynamic stream of just the positive edges (all pairs of vertices that have no edge at the end of the stream have a $(-)$ edge). We further note that our algorithms work even when the complete graph in the end has poly(n)

(+) edges (or (−) edges) between a pair of vertices (the multiple edges are thought of as just one edge and not parallel edges). The space blowup in this case is only a multiplicative factor of $\log n$.

For a pair of vertices, knowing whether it has a (+) or (−) edge at the end of the stream can be done with a 1 bit counter. We just need to remember if the last insertion between that pair of vertices was a (+) or (−) edge. If the stream contains only (+) edges then a counter mod 2 does the job. So in conclusion we just need 1 bit. In the case where we have poly(n) edges between a pair of vertices in the end, we use a counter of size $O(\log n)$ bits.

6.2.4 Standard Technical Tools

Sampling from graph streams. For a graph $G = (V, E^+ \cup E^-)$, it is straightforward to sample an edge uniformly at random in an insertion-only graph stream. For dynamic graphs, the standard tool is the ℓ_0 sampler (Problem 1) that takes $O(\log^3(n))$ space and samples an edge with probability at least $1 - 1/\text{poly}(n)$. However, in our algorithms, we sample *vertices* and store the edges between the sampled vertices. As such, we do *not* have to use the ℓ_0 samplers and we directly keep track of the label of the last update – this only takes $O(1)$ words for each pair of vertices.

Communication complexity and streaming lower bounds. We use the hardness of the Index problem (Definition 2.3.2) from two-party one-way communication complexity to prove the first streaming lower bound. For the second lower bound we use the hardness of Boolean Hidden Hypermatching defined in Section 6.5.1.

6.3 Purely additive approximation

In this section, we give an algorithm for estimating the correlation clustering cost with only additive error (the multiplicative approximation is 1). This idea is based on reducing correlation clustering with δn^2 additive error to an instance of max CSP.

Theorem 10. *There is a (dynamic) streaming algorithm that with high probability gives a* $(1, \delta n^2)$*-approximation for the correlation clustering value and takes space* $O(\log n/\text{poly}(\delta))$ *words and post-processing time* $\log^2 n \cdot (1/\delta)^{O(1/\delta^2 \cdot \log^2 1/\delta)}$.

We start with a claim which shows that restricting to at most $1/\delta$ clusters only gives $O(\delta n^2)$ additional cost.

Claim 6.3.1. *Let* $\mathsf{OPT}_{1/\delta}$ *be the optimal correlation clustering cost when the number of clusters is restricted to at most* $1/\delta$*. Then we have* $\mathsf{OPT} \leqslant \mathsf{OPT}_{1/\delta} \leqslant \mathsf{OPT} + 4\delta n^2$*.*

Proof. Consider the optimal clustering and look at all clusters of size at least $2\delta n$. There are at most $1/2\delta$ such clusters which we keep as is. Now consider all clusters of size $< 2\delta n$ and arbitrarily combine them into clusters with sizes between $2\delta n$ and $4\delta n$. There can be at most $1/2\delta$ such clusters. By merging clusters, we might have increased the cost of negative edges within clusters (observe that merging clusters does not incur any cost from positive edges between them). The increase in cost is at most $(4\delta n)^2/2$ per cluster so at most $4\delta n^2$ over all clusters. This is a specific clustering with at most $1/\delta$ clusters. The optimal clustering on at most $1/\delta$ clusters can only be better, so we get the inequality in the claim. ∎

We now observe that minimizing the cost of disagreement can be done by maximizing the cost of agreement:

Observation 6.3.2. Minimizing the cost of disagreement using at most $1/\delta$ clusters can be done by maximizing the cost of agreement using at most $1/\delta$ clusters and then subtracting that value from $\binom{n}{2}$.

We have now reduced the problem to maximizing the cost of agreement using at most $1/\delta$ clusters. Solving this problem with δn^2 additive error translates to solving the problem of minimizing the cost of disagreement using $1/\delta$ clusters with δn^2 additive error.

We now turn our attention to the Max 2-CSP problem which is defined as follows. A 2-CSP instance $\mathcal{P}$ on n variables $x_1, x_2, \ldots, x_n$ is a set of predicates (or constraints) of the form $P : [q]^2 \to \{0, 1\}$, where every $P = P(x_i, x_j)$ depends only on 2 of the n variables in x. $Var(P)$ denotes the set of variables P is a function of. We think of $\mathcal{P}$ itself as a mapping $\mathcal{P} : [q]^n \to [0, 1]$ defined as $\mathcal{P}(x) = \frac{1}{|\mathcal{P}|} \sum_{P \in \mathcal{P}} P(x)$. The optimum is denoted $\mathsf{OPT}(\mathcal{P}) = \max_{x \in [q]^n} \mathcal{P}(x)$ which is the optimal solution to the Max 2-CSP. For a subset $U \subseteq [n]$, we let $\mathcal{P}_U$ denote the 2-CSP $\mathcal{P}_U = \{P \in \mathcal{P} : Var(P) \subseteq U\}$.

We now show how the problem of maximizing agreement in correlation clustering with at most $1/\delta$ clusters can be formulated as an instance of Max 2-CSP. Consider exactly $q := 1/\delta$ clusters, allowing for the possibility that some clusters remain empty (giving us at most $1/\delta$ true clusters). While the general version of correlation clustering can also be cast as a Max 2-CSP with $q = n$, restricting the number of clusters to $1/\delta$ gives us better parameters.

We associate a variable with each of the n vertices, where each variable takes a value in $[1/\delta]$ indicating the cluster assignment for the corresponding vertex. For each pair of vertices, we define a constraint as follows:

- If the pair of vertices is connected by a *positive edge*, the constraint evaluates to 1 if both vertices are assigned to the *same* cluster (have the same value), and 0 otherwise.
- If the pair of vertices is connected by a *negative edge*, the constraint evaluates to 1 if the vertices are assigned to *different* clusters (have different values), and 0 otherwise.

This defines a Max 2-CSP instance, and our goal is to solve it up to an additive error of δn^2. The following proposition provides an algorithm that achieves this:

Proposition 6.3.3 ([BHHS11]). *Let $\varepsilon > 0$, and let $\mathcal{P}$ be a 2-CSP on n variables over an alphabet of size q such that every variable is in Δ constraints. Let $\varepsilon' > \varepsilon^{-C} \log(q)/\Delta$ for some absolute constant C. Suppose $U \subseteq [n]$ is chosen uniformly at random so that $|U| = \varepsilon' n$. Then,*

$$|\mathbb{E}\left[\mathsf{OPT}(\mathcal{P}_U)\right] - \mathsf{OPT}(\mathcal{P})| \leqslant \varepsilon.$$

Note that in our case, the number of variables is n, the alphabet size is $q = 1/\delta$, and $\delta = n - 1$ since there is a constraint for every pair of variables. We choose $\varepsilon = \delta$ to get an additive error of at most δn^2. This means that we need a uniform sample of $\varepsilon' n$ vertices and have to solve our correlation clustering problem on those vertices. We pick $\varepsilon' n = n \cdot \delta^{-C} \log(1/\delta)/(n-1) \leqslant \mathrm{poly}(1/\delta) := k$, where k is the size of the sample.

Thus, before the stream begins, we sample k vertices uniformly at random and, during the stream, store the subgraph induced on these sampled vertices. After the stream, our goal is to estimate the agreement cost on this subgraph up to an additive error of δk^2, and then scale the result by a factor of n^2/k^2 to approximate the agreement cost on the full graph.

Since runtime is typically not the primary concern in the streaming setting, one naive approach is to solve the problem exactly on the sampled subgraph, which would take $k^k = (1/\delta)^{\mathrm{poly}(1/\delta)}$ time. However, this is an overkill, as we only require an approximate solution with δk^2 additive error. Instead, we aim to apply the best available offline algorithms for estimating the Max-Agree cost on $1/\delta$ clusters with additive error δk^2.

So we can use a PTAS algorithm for max agree on a fixed number of clusters [GG06] and get a $(1-\delta)$ approximation on the sample. Since the max agree cost is always $\Theta(k^2)$ (k is the number of vertices), this is the same as getting an additive approximation of

δk^2. This can be done in time $(1/\delta)^{O(1/\delta^2 \cdot \log^2 1/\delta)}$ with constant probability of success. We can repeat this $O(\log n)$ and output the median which increases the success probability to $1 - 1/\text{poly}(n)$.

Note that the guarantee of [BHHS11] holds only in expectation. Thus, we use parallel repetitions in the stream to guarantee success with high probability instead of just in expectation. To do this we need to repeat the sampling algorithm $r := O(\log n/\delta^2)$ times in parallel and then output the average of all answers. Let $U_1, U_2, \ldots, U_r$ be the samples from the stream and let X_i be $\mathsf{OPT}(\mathcal{P}_{U_i})$. We know that $|\mathbb{E}[\mathsf{OPT}(\mathcal{P}_{U_i})] - \mathsf{OPT}(\mathcal{P})| \leqslant \delta$ for all $i \in [r]$. Let $X = \sum_{i\in[r]} X_i$ be the sum of these values. Since the X_i's are independent and in the range $[0, 1]$, using the Additive Chernoff bound (Proposition 2.4.4) we get:

$$\Pr(|X - \mathbb{E}[X]| > \delta \cdot r) \leqslant 2\exp(-2\delta^2 r^2/r) \leqslant 1/\text{poly}(n).$$

This implies that with high probability, the average value $\bar{X} := X/r$ is at most δ away from its expectation $\mathbb{E}\left[\bar{X}\right] = \mathbb{E}[\mathsf{OPT}(\mathcal{P}_{U_i})]$ for any $i \in [r]$ (this value is the same for all $i \in [r]$). We also know that $|\mathbb{E}[\mathsf{OPT}(\mathcal{P}_{U_i})] - \mathsf{OPT}(\mathcal{P})| \leqslant \delta$ which implies $\left|\bar{X} - \mathsf{OPT}(\mathcal{P})\right| \leqslant 2\delta$.

After scaling up by $\binom{n}{2}$, the difference between $\binom{n}{2} \cdot \bar{X}$ and $\mathsf{OPT}(\mathcal{P})$ is at most $2\delta \cdot \binom{n}{2} \leqslant \delta n^2$ with high probability. Also, the PTAS algorithm approximates X with an additive error of δn^2. Thus, taking a union bound over the failure probabilities of all the copies of the PTAS algorithm and the concentration failing, the value we output has an additive error of at most $O(\delta n^2)$ from the optimal cost of max agree on the original instance with no restriction on the number of clusters (since all the steps only added an error of $O(\delta n^2)$). Finally, we re-scale δ by a constant factor to get at most δn^2 additive error.

The amount of space taken is $O(k^2 \cdot r) = \text{poly}(1/\delta) \cdot \log n$ words because we store r subgraphs on k vertices each. The amount of post-processing time taken is $\log n \cdot (1/\delta)^{O(1/\delta^2 \cdot \log^2 1/\delta)}$ for each of the r copies giving a total runtime of $\log^2 n \cdot (1/\delta)^{O(1/\delta^2 \cdot \log^2 1/\delta)}$. This proves Theorem 10.

For this algorithm, the time complexity of running a PTAS in post-processing is better than running a brute-force algorithm, but is still impractical. To improve the post-processing time complexity we could just estimate the value and not get the entire clustering on the sample, but we are not aware of any better algorithms in the literature that do this. Also, we cannot use any algorithms with a worse multiplicative approximation in post-processing to reduce the time because we are working with maximizing agreement and not minimizing disagreement. Any worse approximation would give us more than δn^2 additive error which is not tolerable.

6.4 A Lower Bound for $O(n^{2-\varepsilon})$ Additive error

In this section, we show that if we *only* allow additive error, any streaming algorithm with poly-logarithm memory cannot cross an error barrier of $\Omega(n^{2-\varepsilon})$ for any constant ε. Here, and throughout, we will refer to this lower bound as the *almost-quadratic lower bound*. The lower bound is weaker than the linear lower bound of Section 6.5 in terms of the multiplicative factor since it only works for $c = 1$. However, it is much stronger in the additive sense: the upper bounds obtained by our algorithms are $O(n^2)$, and the almost-quadratic lower bound matches this term up to an $O(n^{\varepsilon})$ factor – this provides a strong justification of the additive error in our algorithms.

The formal statement of the almost-quadratic lower bound is as follows.

Theorem 11. *There exists a constant C, such that any single-pass streaming algorithm that estimates the optimal value OPT of correlation clustering by a $C \cdot n^{2-\varepsilon}$* purely additive *error (i.e., an estimated value that is at most $(\mathsf{OPT} + C \cdot n^{2-\varepsilon})$) with probability at least $\frac{99}{100}$ has to use a memory of $\Omega(n^{\varepsilon})$ bits, even on labeled complete graphs.*

Note that Theorem 11 does *not* require the stream to be dynamic, which is in contrast to our upper bound results that work for dynamic streams. We obtain the almost-quadratic lower bound by a new reduction from the INDEX problem. On the high level, the instance we construct 'hides' an $\Omega(n^{2-\varepsilon})$ gap between the yes and no cases inside a case-invariant $\Omega(n^2)$ cost. The reduction can be viewed as a more involved variant of the space lower bound for the *exact* correlation clustering in a very recent work [AAD+23]. In a nutshell, we modify their construction to 'boost' the gap between yes and no cases, and apply a new trick to separate the *values* of clustering[2].

We now start the formal reduction proof with the following variant of INDEX.

Problem 7. *Consider a two-player communication problem, where Alice is given a matrix $M \in \{0,1\}$ and Bob is given $i^*, j^* \sim [N] \times [N]$. Alice sends a message to Bob, and Bob outputs the value of $M[i^*, j^*]$.*

Using Proposition 2.3.3, it is trivial to prove that Problem 7 requires $\Omega(N^2)$ bits of communication for Bob to output the correct $M[i^*, j^*]$ with probability at least $\frac{99}{100}$. We shall now construct an instance that creates the desired gap from Problem 7.

[2]Both ideas are discussed separately by [AAD+23], but it was far from clear whether the two strategies can work together to obtain the desired lower bound – this is a main technical work in our proof.

6.4.1 A construction of correlation clustering structural

To continue, we introduce a subroutine for Alice and Bob to construct vertices with a 2-clustering structure, such that for the optimal clustering, Alice's edge will only affect the assignment of one fixed group of vertices. Crucially, the optimal and second-best clustering assignments of the instance only differ on this special group of vertices.

As mentioned, we use a simpler construction in [AAD+23] as our prototype. We start introducing the formal construction with the hard-coded inputs. We insist that the input parameter N is odd in our construction.

Structural hard-coded inputs: An input-invariate sub-graph

1. Both player create $3C$ vertices for each index $i \in [N]$, where C is a parameter.
2. Connect each collection of C vertices with $(+)$ edges, i.e. make them cliques of size C.
3. For each index i, put the 3 corresponding collections into 3 groups: L, $R^{(1)}$, and $R^{(2)}$. We let $L(i)$ denote the collection of vertices correspond to (i) in L, and define $R^{(1)}(i)$ and $R^{(2)}(i)$ analogously.
4. Create $100 \cdot CN$ vertices and name this new group V. Divide V into two equal-size partitions of size $50 \cdot CN$ each, and name them $V_\uparrow$ and $V_\downarrow$.
5. Add $(+)$ edges between all vertex pairs inside $V_\uparrow \times V_\uparrow$ and inside $V_\downarrow \times V_\downarrow$; add $(-)$ edges between vertex pairs of $V_\uparrow \times V_\downarrow$.

With the structural hard-coded inputs, we can give our complete construction as follows.

Dense-Two-Clusters: **A family of graphs from INDEX**

1. Add the structural hard-coded inputs, prescribed as below, to obtain L, $R^{(1)}$, $R^{(2)}$, $V_\uparrow$, $V_\downarrow$, and the corresponding edges.
2. Alice edges:
   (a) For each index (i, j), Alice adds $2 \cdot C^2$ edges as follows:

- If $M[i,j] = 0$, Alice adds $(+)$ edges between every vertex pairs in $L(i)$ and $R^{(1)}(j)$, and $(-)$ edges between every vertex pairs in $L(i)$ and $R^{(2)}(j)$.
- Otherwise, if $M[i,j] = 1$, Alice adds $(+)$ edges between every vertex pairs in $L(i)$ and $R^{(2)}(j)$, and $(-)$ edges between every vertex pairs in $L(i)$ and $R^{(1)}(j)$.

3. Bob edges:

   (a) For every index other than i^*, Bob divides the vertices in L to $L_\uparrow$ and $L_\downarrow$, such that
   - All vertices inside $L_\uparrow$ and inside $L_\downarrow$ are connected with $(+)$ edges; all vertex pairs in $L_\uparrow \times L_\downarrow$ are connected with $(-)$ edges.
   - Vertices in $L_\uparrow$ are connected to *all vertices* in $V_\uparrow$ with $(+)$ edges and to *all vertices* in $V_\downarrow$ with $(-)$ edges.
   - Vertices in $L_\downarrow$ are connected to *all vertices* in $V_\downarrow$ with $(+)$ edges and to *all vertices* in $V_\uparrow$ with $(-)$ edges.

   (b) For every index other than j^*, Bob divides the vertices in $R^{(1)}$ to $R^{(1)}_\uparrow$ and $R^{(1)}_\downarrow$ and $R^{(2)}$ to $R^{(1)}_\uparrow$ and $R^{(2)}_\downarrow$ with the *same* lexigraphical order.

   (c) Bob connects all vertices in $R^{(1)}(j^*)$ with $(+)$ edges to $V_\downarrow$, $R^{(1)}_\downarrow$, and $R^{(2)}_\downarrow$, and $(-)$ edges to $V_\uparrow$, $R^{(1)}_\uparrow$, and $R^{(2)}_\uparrow$.

   (d) Bob connects all vertices in $R^{(2)}(j^*)$ with $(+)$ edges to $V_\uparrow$, $R^{(1)}_\uparrow$ and $R^{(2)}_\uparrow$, and $(-)$ edges to $V_\downarrow$, $R^{(1)}_\downarrow$, $R^{(2)}_\downarrow$.

   (e) Bob connects vertices in $L(i^*)$ with $(+)$ edges to every vertices other than $R^{(1)}$ and $R^{(2)}$.

   (f) Bob connect $(-)$ edges between all vertex pairs in $(L_\uparrow \times R^{(1)}_\downarrow)$, $(L_\uparrow \times R^{(2)}_\downarrow)$, $(L_\downarrow \times R^{(1)}_\uparrow)$, and $(L_\downarrow \times R^{(2)}_\uparrow)$; furthermore, Bob connects $(+)$ edges between all vertex pairs in $(L_\uparrow \times R^{(1)}_\uparrow)$, $(L_\uparrow \times R^{(2)}_\uparrow)$, $(L_\downarrow \times R^{(1)}_\downarrow)$, and $(L_\downarrow \times R^{(2)}_\downarrow)$.

It is straightforward to verify that the graph constructed by Alice and Bob is complete and every edge has a label of either $(+)$ or $(-)$. In what follows, we prove that Dense-Two-Clusters has a neat structure that the optimal clustering always contains ex-

actly 2 clusters, and the optimal and second-optimal clusterings differ only by the cluster of $L(i^*)$. More concretely, we have the following lemma.

Lemma 6.4.1. *The optimal clustering of* Dense-Two-Clusters *contains two clusters:*

- *The first cluster (the* up cluster*) contains the vertices of* $L_\uparrow, V_\uparrow, R^{(1)}_\uparrow, R^{(2)}_\uparrow$ *(including* $R^{(2)}(j^*)$*);*
- *The second cluster (the* down cluster*) contains the vertices of* $L_\downarrow, V_\downarrow, R^{(1)}_\downarrow, R^{(2)}_\downarrow$ *(including* $R^{(1)}(j^*)$*).*

Furthermore, the remaining vertex $L(i^*)$ *is in the up cluster if* $M[i^*, j^*] = 1$*, and is in the down cluster if* $M[i^*, j^*] = 0$*. The second-optimal clustering differs from the optimal clustering by switching* $L(i^*)$ *to the other cluster, and the cost differs by 2.*

Lemma 6.4.1 was originally proved by [AAD+23]. We give a self-contained proof in our paper, and our proof strategy turns out to be technically more applicable for our purposes in later steps. To begin with, we define the notion of up vertices and down vertices by their edge connectivity to $V_\uparrow$ as follows.

Definition 6.4.2 (Up and down vertices)**.** *Fix a graph* $G = (L \cup R^{(1)} \cup R^2 \cup V, E)$ *sampled from* Dense-Two-Clusters*, we say a vertex* $v \neq L(i^*)$ *is an* up vertex *if and only if* $|E^+(v, V_\uparrow)| \geqslant |E^-(v, V_\uparrow)|$*, i.e. among the vertices in* $V_\uparrow$*,* v *has more* $(+)$ *edges in* $V_\uparrow$ *than* $(-)$ *edges. We say* v *is a* down vertex *otherwise.*

Note that the above definition also covers $R^{(1)}(j^*)$ as a down vertex and $R^{(2)}(j^*)$ as an up vertex. Intuitively, a vertex v becomes an up vertex if it is densely connected to the 'up side' with $(+)$ edges, and vice versa. The up and down vertices have very 'local' properties: it has many $(+)$ edges to one side and many $(-)$ edges to the other. More formally, we characterize this property as the following condition.

Condition 1. We say two groups of vertices, namely the up and down vertices, satisfy the Cond* condition if there exists a global constant N such that:

1. For up-up and down-down vertex pairs, there are
   (a) Every up vertex has no $(-)$ edges and at least $50N$ $(+)$ edges connecting to vertices in $R^{(1)}_\uparrow, R^{(2)}_\uparrow, V_\uparrow$, and at most N $(-)$ edges connecting other up vertices.

(b) Every down vertex has no $(-)$ edges and at least $50N$ $(+)$ edges connecting to vertices in $R^{(1)}_{\downarrow}, R^{(2)}_{\downarrow}, V_{\downarrow}$, and at most N $(-)$ edges connecting other down vertices.

2. For up-down vertex pairs, there are

   (a) Every up vertex has no $(+)$ edges and at least $50N$ $(-)$ edges connecting to vertices in $R^{(1)}_{\downarrow}, R^{(2)}_{\downarrow}, V_{\downarrow}$, and at most N $(+)$ edges connecting other down vertices.

   (b) Every down vertex has no $(+)$ edges and at least $50N$ $(-)$ edges connecting to vertices in $R^{(1)}_{\uparrow}, R^{(2)}_{\uparrow}, V_{\uparrow}$ and at most N $(+)$ edges connecting other up vertices.

We observe that the graphs sampled from Dense-Two-Clusters satisfy the Cond*.

Observation 6.4.3. Cond* in Condition 1 holds in any graph sampled from Dense-Two-Clusters.

We now use Condition 1 and Observation 6.4.3 to characterize the structure of the optimal clustering. In particular, we first prove that with Condition 1, there can be at most one cluster consisting of only up vertices and similarly for down vertices.

Claim 6.4.4. *Assuming* Cond* *in Condition 1, there can be at most one cluster consisting of only up vertices (similarly for down vertices).*

Proof. Assume towards a contradiction that there are clusters C_1 and C_2 consisting of only up vertices. Create a new clustering by merging C_1 and C_2 into a new cluster C. We observe that merging into C does not change the cost induced by the edges with an endpoint in $V - C$. By line 1 of Condition 1, by merging C_1 and C_2 into C, the cost of the new clustering strictly decreases. Thus, the original clustering was not optimal giving a contradiction. Therefore, there can be at most one cluster consisting of only up vertices. The proof for down vertices is analogous. ∎

Using Claim 6.4.4 together with Condition 1, we can show that any optimal clustering cannot contain put up and down vertices into the same cluster.

Claim 6.4.5. *Assuming* Cond* *in Condition 1, a cluster in the optimal clustering cannot contain both up vertices and down vertices.*

Proof. We first prove the statement for vertices among $V \cup R^{(1)} \cup R^{(2)}$ and vertices among L, respectively. Assume towards a contradiction that there is a cluster C that contains both up vertices and down vertices among $V \cup R^{(1)} \cup R^{(2)}$. Create a new clustering by splitting C into clusters $C_{\uparrow} - L$ containing the up vertices and $C_{\downarrow} - L$ containing the down

vertices. We observe that splitting C does not change the cost induced by the edges with an endpoint in $V - C$. By line 2 of Condition 1, there is a $(-)$ edge from every vertex in $C_\uparrow - L$ to $C_\downarrow - L$ (for vertices in $V \cup R^{(1)} \cup R^{(2)}$), and the cost of the new clustering strictly decreases. Thus, the original clustering was not optimal giving a contradiction. Therefore, a cluster in the optimal clustering cannot contain both up vertices and down vertices for vertices in $V \cup R^{(1)} \cup R^{(2)}$. With the same argument as above, we can prove that the statement holds for vertices among L.

We now proceed to prove the statement for all vertices. By Claim 6.4.4 and the result on vertices among $V \cup R^{(1)} \cup R^{(2)}$, the vertices among $V_\uparrow \cup R^{(1)}_\uparrow \cup R^{(2)}_\uparrow$ must be clustered together, and we let this cluster be $C_\uparrow - L$. We assume towards a contradiction that there is a cluster C that contains both up vertices and down vertices among $L_\downarrow \cup C_\uparrow - L$. As such, we can split C into clusters $C_\uparrow - L$ and vertices in $L_\downarrow$ without affecting the cost induced by the edges with an endpoint in $V - C$. Fix a vertex $v \in L_\downarrow$, although there might be at most N $(+)$ edges between v and $C_\uparrow - L$, there are at least $50N$ $(-)$ edges that leads to a strict cost decrement by this split. As such, we can obtain a contradiction, which proves that all up and down vertices cannot stay in the same cluster in the optimal clustering. ∎

Using Claim 6.4.4, Claim 6.4.5 together with Observation 6.4.3, we can conclude that the optimal clustering for Dense-Two-Clusters has at most three clusters; one containing $L(i^*)$ (C_{i^*}), another containing only up vertices ($C_\uparrow := L_\uparrow \cup V_\uparrow \cup R^{(1)}_\uparrow \cup R^{(2)}_\uparrow$) and the last containing only down vertices ($C_\downarrow := L_\uparrow \cup V_\downarrow \cup R^{(1)}_\downarrow \cup R^{(2)}_\downarrow$). We will now use the edges of $L(i^*)$ to conclude that $L(i^*)$ must be clustered together with either $C_\uparrow$ or $C_\downarrow$.

Claim 6.4.6. *There are exactly* 2 *clusters in the optimal clustering, and vertex* $L(i^*)$ *must be clustered either with* $C_\uparrow$ *or* $C_\downarrow$.

Proof. We first argue that the number of clusters is > 1. Consider that there is just one cluster C. Remove all the up vertices and move them to a new cluster $C_\uparrow$. The decrease in the cost is at least $(50N)^2$ because of the non-edges between $V_\uparrow$ and $V_\downarrow$. The increase in the cost is at most $103N$ because of the edges between i^* and the up vertices. There is a net decrease in cost and thus, the number of clusters is > 1. We now assume towards a contradiction that there are 3 clusters $C_{i^*}, C_\uparrow, C_\downarrow$.

We first consider the case where C_{i^*} contains only vertex i^*. Move i^* to $C_\uparrow$. The increase in the cost is at most N due to the non-edges to some up vertices. But the decrease in the cost is at least $50N$ because of edges to $V_\uparrow$. Thus, we get a contradiction in this case.

We now consider the case where C_{i^*} also contains vertices other than i^*. Let there be u up vertices and d down vertices in C_{i^*} and wlog assume $u > 0$. Move the up vertices in C_{i^*} to $C_\uparrow$. The decrease in cost is $u \cdot d + u \cdot |C_\uparrow|$ where the first term is from the non-edges between the up and down vertices in C_{i^*} and the second term is from the edges between the up vertices in C_{i^*} and $C_\uparrow$. The increase in cost is at most u which is only due to edges between i^* and the up vertices. So there the net decrease in cost is $u \cdot d + u \cdot |C_\uparrow| - u > 0$ (even when $d = 0$). ∎

Finalizing the proof of Lemma 6.4.1. By Claim 6.4.4, Claim 6.4.5, and Claim 6.4.6, there are exactly two clusters $C_\uparrow$ (containing all up vertices) and $C_\downarrow$ (containing all down vertices) and i^* is a part of one of these depending on its edges to $j^*_\uparrow$ and $j^*_\downarrow$. As such, we can observe $L(i^*)$ joins $C_\uparrow$ if it has $(+)$ edge to $R^{(2)}_\uparrow(j^*)$ and $(-)$ edge to $R^{(1)}_\uparrow(j^*)$, and vice versa. The two scenarios corresponds exactly to $M[i^*, j^*] = 1$ and $M[i^*, j^*] = 0$. Finally, note that if we move the assignment of $L(i^*)$, the cost increases exactly 2, while the cost increases by at least $48N$ if we move any other vertex. As such, the second-best clustering is to keep $C_\uparrow$ and $C_\downarrow$, and switch $L(i^*)$ to the other cluster.

6.4.2 Multi-copy correlation clustering structure

We now slightly tweak our original model by introducing 'duplicates' for each vertex in the Dense-Two-Clusters distribution. In particular, for each vertex in L, $R^{(1)}$, $R^{(2)}$ and V, we make it as a group of K vertices for some $K \geqslant 1$ which we specify later. In other works, for each $L(i)$ (resp. $R^{(1)}(i)$, $R^{(2)}(i)$, $V(i)$) in the Dense-Two-Clusters family, we have a group $\mathcal{L}(i)$ (resp. $\mathcal{R}^{(1)}(i)$, $\mathcal{R}^{(2)}(i)$, $\mathcal{V}(i)$) of K vertices. Inside each group of vertices, we connect the vertices as a clique of $(+)$ edges; between different groups, we add K^2 edges with the same label as the single labeled edge in Dense-Two-Clusters. We will eventually argue that the best clustering of such a family remains a 2-cluster structure, and the cost gap between the best and the second-best clustering becomes $O(K)$.

We now give a formal description of the augmented family of instances as follows.

Duplicate-Dense-Two-Clusters(K)**: A family of graphs from INDEX**

1. Alice and Bob samples a graph G' from Dense-Two-Clusters.
2. For each vertex $L(i)$, the players make a group $\mathcal{L}(i)$ of K copies of vertices. In the same manner, they make each vertex in $R^{(1)}(i)$, $R^{(2)}(i)$ and $V(i)$ to groups $\mathcal{R}^{(1)}(i)$, $\mathcal{R}^{(2)}(i)$ and $\mathcal{V}(i)$ with K vertices in each group.

3. Fix any group $\mathcal{A}$ and two vertices $(u, v) \in \mathcal{A}$, add a $(+)$ edge between u and v.

4. For any two groups $\mathcal{A}$ and $\mathcal{B}$ such that $u \in \mathcal{A}$ and $v \in \mathcal{B}$, add an edge whose label is consistent with the edge between the two vertices in G' that induces the groups.

Let G be the resulting group sampled from Duplicate-Dense-Two-Clusters(K), we now prove that the new family of instances retains the two-cluster structure. To proceed formally, we introduce new notation of $\mathcal{L}$, $\mathcal{R}^{(1)}$, $\mathcal{V}$, $\mathcal{R}^{(2)}$ to denote the set of the vertices of all groups induced by the vertices in L, $R^{(1)}$, V, and $R^{(2)}$. In the same manner, we use $\mathcal{L}_\uparrow$, $\mathcal{R}^{(1)}_\uparrow$, $\mathcal{V}_\uparrow$, $\mathcal{R}^{(2)}_\uparrow$ (resp. $\mathcal{L}_\downarrow$, $\mathcal{R}^{(1)}_\downarrow$, $\mathcal{V}_\downarrow$, $\mathcal{R}^{(2)}_\downarrow$) to denote the vertices in the groups induced by $L_\uparrow, V_\uparrow, R^{(1)}_\uparrow, R^{(2)}_\uparrow$ (resp. $L_\downarrow, V_\downarrow, R^{(1)}_\downarrow, R^{(2)}_\downarrow$). The structure of the optimal clustering can be characterized as follows.

Lemma 6.4.7. *The optimal clustering of* Duplicate-Dense-Two-Clusters(K) *for any integer* $K \geqslant 1$ *contains two clusters:*

- *The first cluster (the* up cluster*) contains the vertices of* $\mathcal{L}_\uparrow, \mathcal{V}_\uparrow, \mathcal{R}^{(1)}_\uparrow, \mathcal{R}^{(2)}_\uparrow$ *(including* $\mathcal{R}^{(2)}(j^*)$*);*
- *The second cluster (the* down cluster*) contains the vertices of* $\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}^{(1)}_\downarrow, \mathcal{R}^{(2)}_\downarrow$ *(including* $\mathcal{R}^{(1)}(j^*)$*).*

Furthermore, all of the remaining vertices of $\mathcal{L}(i^*)$ *is in the up cluster if* $M[i^*, j^*] = 1$*, and is in the down cluster if* $M[i^*, j^*] = 0$*.*

Towards the proof of Lemma 6.4.7, we first show that the optimal clustering for Duplicate-Dense-Two-Clusters(K) follows the same two-cluster structure on $(\mathcal{L}_\uparrow, \mathcal{V}_\uparrow, \mathcal{R}^{(1)}_\uparrow, \mathcal{R}^{(2)}_\uparrow)$ and $(\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}^{(1)}_\downarrow, \mathcal{R}^{(2)}_\downarrow)$ as happened for Dense-Two-Clusters.

Claim 6.4.8. *In the optimal clustering, there exists a cluster* C_1 *that contains all vertices in* $\mathcal{L}_\uparrow, \mathcal{V}_\uparrow, \mathcal{R}^{(1)}_\uparrow, \mathcal{R}^{(2)}_\uparrow$ *and a cluster* C_2 *that contains all vertices in* $\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}^{(1)}_\downarrow, \mathcal{R}^{(2)}_\downarrow$*.*

Proof. The claim is a natural generalization of Claim 6.4.4 and Claim 6.4.5. Similar to Definition 6.4.2, we define the up vertices as the vertices (except from the group $\mathcal{L}(i^*)$) whose $(+)$ edges to $\mathcal{L}_\uparrow$ is more than the $(-)$ edges. Note that by copying each vertex K times and adding edges with the rules in Lines 3 and 4, we never add $(+)$ edges between

the up vertices and down vertices among the vertices in $\mathcal{R}^{(1)}$, $\mathcal{R}^{(2)}$, and $\mathcal{V}$. Furthermore, for each vertex u in $\mathcal{L}(i)$ for any $i \neq i^*$, suppose w.log. that the vertex is an up vertex, we add at most $K \cdot N$ $(+)$ edges to other down vertices and at most $K \cdot N$ $(-)$ edges to other up vertices. As such, Cond^* in Condition 1 holds with global constant $K \cdot N$. As such, we can apply Claim 6.4.4 and Claim 6.4.5 to obtain the desired claim. ∎

We now need to handle the clustering of the vertices among the group of $\mathcal{L}(i^*)$, $\mathcal{R}^{(1)}(j^*)$ and $\mathcal{R}^{(2)}(j^*)$. We show that the vertices should be clustered together rather than being split apart, which is sufficient to prove Lemma 6.4.7 by further applying Claim 6.4.6.

Claim 6.4.9. *In the optimal clustering of* Duplicate-Dense-Two-Clusters(K) *for any integer* $K \geqslant 1$*, all vertices inside* $\mathcal{L}(i^*)$ *are in the same cluster.*

Proof. Assume towards a contradiction that there are at least 2 clusters with vertices from $\mathcal{L}(i^*)$ (in the optimal clustering). Consider clusters C_1 and C_2 with multiple vertices $u_1, u_2, \cdots, u_m$ from $\mathcal{L}(i^*)$. Let I_1 be all vertices from $\mathcal{L}(i^*)$ in cluster C_1 and let R_1 be the remaining vertices in C_1. Similarly, define I_2 and R_2 as the vertices in cluster C_2 that are inside and outside $\mathcal{L}(i^*)$. We observe that if we move vertices between C_1 and C_2 the cost to vertices with endpoint in $V - (C_1 \cup C_2)$ remains unchanged. Consider any vertex u in $\mathcal{L}(i^*)$ and calculate the cost c_1 of clustering it in C_1 with respect to R_1, R_2, i.e. the number of $(+)$ edges to R_2 plus the number of $(-)$ edges to R_1. Similarly calculate the cost c_2 of clustering it in C_2 with respect to R_1, R_2. Suppose w.log. that $c_1 \leqslant c_2$, we can create a new clustering by moving all copies of i^* in C_2 to C_1. Since $c_1 \leqslant c_2$ the cost to $R_1 \cup R_2$ can only decrease (for each u of $\mathcal{L}(i^*)$ moved). Also, the cost induced by the vertex pairs $u_1 \in I_1$ and $u_2 \in I_2$ for any u_1 and u_2 goes to 0. Thus, the clustering cost strictly decreases, which forms a contradiction to the optimality of the clustering. ∎

Finalizing the proof of Lemma 6.4.7. By Claim 6.4.8, there are exactly two clusters $C_\uparrow$ (containing all up vertices) and $C_\downarrow$ (containing all down vertices) and *all* vertices in $\mathcal{L}(i^*)$ are part of one of the clusters depending on its edges to $j^*_\uparrow$ and $j^*_\downarrow$. As such, we can observe vertices of $\mathcal{L}(i^*)$ join $C_\uparrow$ if it has $(+)$ edge to $R^{(2)}_\uparrow(j^*)$ and $(-)$ edge to $R^{(1)}_\uparrow(j^*)$, and vice versa. As such the optimal clustering structure as prescribed in Lemma 6.4.7 is obtained.

6.4.3 Cost Testing Lower Bound – A Variate of the Hard Instance

So far, we have not discussed the separation of costs in the yes and no instances. Indeed, one can observe that although the optimal clustering changes in a regulated manner depending

on the values of $M[i^*, j^*]$, the *cost* of the optimal clusters are the same. We now apply a trick to 'tweak' the construction in Section 6.4.1 such that the clustering structure remains except vertices in the special group (i^*, j^*). Such a trick was first employed in [AAD+23]; however, the trick was only applied to a single vertex in their case, and the analysis is much more straightforward. Our technical contribution here is to show that the trick applies even with multiple copies of vertices in each group.

We now introduce the value-testing trick as an algorithm that given an oracle for correlation clustering with $o(n^{2-\Omega(1)})$ (purely) additive error, returns the correct answer of INDEX:

CC-INDEX-ALG: An algorithm for INDEX given Tool-CC(ε) – an oracle that returns the optimal cost of correlation clustering with $(1, \frac{1}{10^6} \cdot n^{2-\varepsilon})$ error

1. Alice and Bob construct a graph G from Duplicate-Dense-Two-Clusters(K) with $K = N^{\frac{1}{\varepsilon}-1}$ for arbitrarily small $\varepsilon > 0$.
2. The players run two subroutines:
   (a) In the first subroutine, the players run **Tool-CC**(ε) on graph G, and obtain CC_1.
   (b) In the second subroutine, the players run **Tool-CC**(ε) on graph G' as follows:
      i. Arbitrarily pick $10CN$ vertices from $\mathcal{V}_\downarrow$ on G, and name the set as $\mathcal{V}'_\downarrow$.
      ii. For each vertex $u \in \mathcal{L}(i^*)$, change the $(+)$ edge between u and $\mathcal{V}'_\downarrow$ to a $(-)$ edge (note that Bob exclusively controls all edges incident on $\mathcal{V}$ so he can do this).
      iii. The players obtain the cost CC_2.
3. If $\mathrm{CC}_1 - \mathrm{CC}_2 \geqslant 10K^2N - \frac{1}{25}K^2$, return $M[i^*, j^*] = 1$; otherwise, return $M[i^*, j^*] = 0$.

We will eventually show that CC-INDEX-ALG succeeds with the same probability as **Tool-CC**(ε), which is sufficient to establish the reduction and prove Theorem 11. To this end, we will prove the optimal clustering structure for both subroutines in both cases, and show that the cost gap is greater than $10K^2N - \frac{1}{25}K^2$ if and only if $M[i^*, j^*] = 1$. We start with observing the basic clustering structure is preserved in G' for the second subroutine.

Observation 6.4.10. In the optimal clustering for both G and G', there exists a cluster

C_1 that contains all vertices in $\mathcal{L}_\uparrow, \mathcal{V}_\uparrow, \mathcal{R}_\uparrow^{(1)}, \mathcal{R}_\uparrow^{(2)}$ and a cluster C_2 that contains all vertices in $\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}_\downarrow^{(1)}, \mathcal{R}_\downarrow^{(2)}$, and the vertices in $\mathcal{L}(i^*)$ are in the same clusters.

Proof. The property holds for G simply by Claim 6.4.8 and Claim 6.4.9. Furthermore, only edges incident on $\mathcal{L}(i^*)$ change in G', which does not affect the correctness of Claim 6.4.8. Finally, we never used any property for edges with endpoint(s) outside $\mathcal{L}(i^*)$, $\mathcal{R}^{(1)}(j^*)$ and $\mathcal{R}^{(2)}(j^*)$ in the proof of Claim 6.4.9; as such, the property in Claim 6.4.9 still holds. ∎

Based on Observation 6.4.10, we now establish the optimal clustering for the graph G' in the second subroutine for both the yes and no cases. We can then analyze the cost difference therein.

Lemma 6.4.11. *The optimal clustering of G' for $M[i^*, j^*] = 0$ contains two clusters:*

- *The first cluster (the* up cluster*) contains the vertices of $\mathcal{L}_\uparrow, \mathcal{V}_\uparrow, \mathcal{R}_\uparrow^{(1)}, \mathcal{R}_\uparrow^{(2)}$ (including $\mathcal{R}^{(2)}(j^*)$) and $\mathcal{L}(i^*)$;*
- *The second cluster (the* down cluster*) contains the vertices of $\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}_\downarrow^{(1)}, \mathcal{R}_\downarrow^{(2)}$ (including $\mathcal{R}^{(1)}(j^*)$).*

Furthermore, the optimal clustering cost of G' (CC_2) decreases by $(10K^2N - 2K^2)$ in comparison to the optimal clustering cost of G (CC_1).

Proof. We first prove the clustering structure. By Observation 6.4.10, the only undecided clustering is the cluster that contains $\mathcal{L}(i^*)$. There are three possible cases: $\mathcal{L}(i^*)$ forms an individual cluster, $\mathcal{L}(i^*)$ is clustered with $C_1 = \mathcal{L}_\uparrow \cup \mathcal{V}_\uparrow \cup \mathcal{R}_\uparrow^{(1)} \cup \mathcal{R}_\uparrow^{(2)}$, and $\mathcal{L}(i^*)$ is clustered with $C_1 = \mathcal{L}_\downarrow \cup \mathcal{V}_\downarrow \cup \mathcal{R}_\downarrow^{(1)} \cup \mathcal{R}_\downarrow^{(2)}$. The only part that induces variate costs between the 3 clustering schemes are the edges with exactly one endpoint in $\mathcal{L}(i^*)$. As such, it suffices to choose the optimal scheme that minimizes the costs induced by these edges.

If the vertices of $\mathcal{L}(i^*)$ form an individual cluster, there are at least $90KN$ $(+)$ edges from every vertex in $\mathcal{L}(i^*)$ to $\mathcal{V}$. As such, the cost induced by edges with exactly one endpoint in $\mathcal{L}(i^*)$ is at least $90K^2N$. On the other hand, if $\mathcal{L}(i^*)$ joins the cluster C_2, each vertex in $\mathcal{L}(i^*)$ has at least $50KN$ $(+)$ edges to C_1 and $10KN$ $(-)$ edges in C_2, which induces a cost of at least $60K^2N$. Finally, if $\mathcal{L}(i^*)$ joins the cluster C_1, it has at most $40KN$ $(+)$ between $\mathcal{V}_\downarrow$. Furthermore, it has at most $3KN$ other edges that can possibly induce any cost. This makes the induced cost to be at most $43K^2N$. As such, clearly the cluster that contains $\mathcal{L}(i^*)$ should join cluster C_1.

We now analyze the optimal cost of correlation clustering on G' compared to CC_1 on G. Since only the clustering of $\mathcal{L}(i^*)$ changes, it suffices to analyze the cost difference induced by the edges with exactly one endpoint in $\mathcal{L}(i^*)$. We first observe that the costs induced by the following set of edges do *not* change between CC_1 and CC_2:

1. Edges between $\mathcal{L}(i^*)$ and $\mathcal{L}(j)$ for any $j \neq i^*$;
2. Edges between $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)} - \mathcal{R}^{(1)}(j^*)$;
3. Edges between $\mathcal{L}(i^*)$ and $\mathcal{R}^{(2)} - \mathcal{R}^{(2)}(j^*)$.

As such, we only analyze the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$, $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)}(j^*)$, $\mathcal{L}(i^*)$ and $\mathcal{R}^{(2)}(j^*)$. In the optimal clustering of G, the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$ always induce a cost of $50K^2N$. On the other hand, in the optimal clustering of G', the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$ induce a cost of $40K^2N$ since there are $10K^2N$ $(-)$ edges that do not induce cost. As such, among the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$, the CC_2 is $10K^2N$ smaller than CC_1.

We now look at the edges between $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)}(j^*), \mathcal{R}^{(2)}(j^*)$. Note that in the optimal clustering of G, these edges induce 0 all $(+)$ edges are in the same cluster and all $(-)$ edges are in different ones. On the other hand, for the optimal clustering on G', each set of edges will induce a cost of K^2. Therefore, among the edges between $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)}(j^*), \mathcal{R}^{(2)}(j^*)$, the CC_2 is $2K^2$ larger than CC_1.

Summing up the above terms gives us the conclusion that the cost difference between CC_2 and CC_1 is $10K^2N - 2K^2$. ∎

From Lemma 6.4.11, we can see that the clustering cost reduction comes from the $(-)$ edges between $\mathcal{L}(i^*)$ and $\mathcal{V}'_{\downarrow}$. However, since the optimal clustering of G' 'switches' the partition of $\mathcal{L}(i^*)$, it further incurs some increment of the cost. In contrast, we will show next that in the yes case, i.e. $M[i^*, j^*] = 1$, the optimal clustering remains the same as in G. As such, the optimal clustering does *not* pay the extra cost induced by the 'switching' of vertices in $\mathcal{L}(i^*)$. The formal statement of the above intuition is as follows.

Lemma 6.4.12. *The optimal clustering of G' for $M[i^*, j^*] = 1$ is the same as the optimal clustering of G, i.e.:*

- *The first cluster (the* up cluster*) contains the vertices of $\mathcal{L}_{\uparrow}, \mathcal{V}_{\uparrow}, \mathcal{R}^{(1)}_{\uparrow}, \mathcal{R}^{(2)}_{\uparrow}$ (including $\mathcal{R}^{(2)}(j^*)$) and $\mathcal{L}(i^*)$;*

- *The second cluster (the* down cluster*) contains the vertices of* $\mathcal{L}_\downarrow, \mathcal{V}_\downarrow, \mathcal{R}^{(1)}_\downarrow, \mathcal{R}^{(2)}_\downarrow$ *(including* $\mathcal{R}^{(1)}(j^*)$*).*

Furthermore, the optimal clustering cost of G' *(*CC_2*) decreases by* $10K^2N$ *in comparison to the optimal clustering cost of* G *(*CC_1*).*

Proof. We define C_1 and C_2 in the same way as in Lemma 6.4.11, and the clustering structure proof follows from exactly the same argument as the proof of Lemma 6.4.11. We omit the details to avoid excessive repetition.

We now analyze the optimal cost of correlation clustering on G' compared to CC_1 on G. Again, the same as the analysis of Lemma 6.4.11, we only need to analyze the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$, $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)}(j^*)$, $\mathcal{L}(i^*)$ and $\mathcal{R}^{(2)}(j^*)$.

Among the edges between $\mathcal{L}(i^*)$ and $\mathcal{V}$, we can obtain that CC_2 is $10K^2N$ smaller than CC_1 with exactly the same argument as in Lemma 6.4.11. Furthermore, for the edges between $\mathcal{L}(i^*)$ and $\mathcal{R}^{(1)}(j^*), \mathcal{R}^{(2)}(j^*)$, note that the cluster of $\mathcal{L}(i^*)$ does *not* change between the optimal clusterings of G and G', as such, the cost induced by these edges stays 0.

Summing up the above terms gives us the conclusion that the cost difference between CC_2 and CC_1 is $10K^2N$. ∎

Finalizing the proof of Theorem 11. If the oracle for correlation clustering has additive error of $\frac{1}{10^6} \cdot n^{2-\varepsilon}$ and no multiplicative error, the cost estimation errors in both subroutines are at most $\frac{103^2}{10^6} \cdot K^{2-2\varepsilon} N^{2-2\varepsilon} \leqslant \frac{1}{50} K^2$. As such, if $M[i^*, j^*] = 0$, the estimation gap between CC_1 and CC_2 is at most $10K^2N - \frac{24}{25}K^2 < 10K^2N - \frac{1}{25}K^2$. On the other hand, if $M[i^*, j^*] = 1$, the estimation gap between CC_1 and CC_2 is at least $10K^2N - \frac{1}{25}K^2$, as desired.

Finally, the communication complexity of the problem is $\Omega(N^2)$, and we have $n = 103KN = O(N^{1/\varepsilon})$. As such, any such **Tool-CC**(ε) must have a communication complexity of $\Omega(n^{2\varepsilon})$ bits, which implies the desired $\Omega(n^\varepsilon)$ space lower bound for the streaming algorithms.

6.5 A Lower Bound for $O(n)$ Additive error

In this section, we show that any insertion-only streaming algorithm that gets a $(c, \varepsilon n)$-approximation for $c < 1.2$ and $O(n)$ additive error needs $\Omega(\sqrt{n})$ bits of space. Here, and throughout, we will call this lower bound the *linear lower bound*. Formally, we have:

Theorem 12. *Let $c \in [1, \frac{6}{5})$ and $\varepsilon \in (0, \frac{6}{5} - c)$. Any single-pass streaming algorithm that estimates the optimal value OPT of correlation clustering by a $(c, \varepsilon n)$-approximation (i.e., an estimated value that is at most $(c \cdot \mathsf{OPT} + \varepsilon \cdot n)$) with probability at least $\frac{99}{100}$ has to use a memory of $\Omega(\sqrt{n})$ bits, even on labeled (complete) graphs.*

Similar to the case in Theorem 11, Theorem 12 does *not* require the stream to be dynamic. Compared with the *almost-quadratic lower bound* we show in Section 6.4, the lower bound in Theorem 12 is weaker in the additive sense. However, it allows the multiplicative approximation of the algorithm to be > 1, while the lower bound in Section 6.4 only rules out algorithms with purely additive errors.

6.5.1 Gap Cycle Counting with Odd Cycles

Our lower bound uses the celebrated machinery from Boolean Hidden Hypermatching (BHH) and Gap Cycle Counting (GCC) pioneered by [VY11]. The Gap Cycle Counting (GCC) lower bound states that for any algorithm to distinguish whether a graph consists of cycles with length $2t$ or cycles with length $4t$ for some $t \geqslant 2$, a memory of $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits is necessary. On a high level, our plan is to show that for graphs similar to the ones prescribed in the GCC problem, the values of correlation clustering are different by an additive gap of $O(n)$. Therefore, by a reduction argument, any algorithm that breaks this barrier of additive gap requires $\omega(\text{polylog}\,(n))$ memory.

The roadblock here is that the original version of Gap Cycle Counting, which only supports *even* cycles, is not sufficient to distinguish the values for correlation clustering. In fact, one can show that for graphs consisting of cycles with any *even* length, the optimal cost for correlation clustering is always $\frac{n}{2}$. To overcome this challenge, we modify the reduction from Boolean Hidden Hypermatching to obtain new hardness results on Gap Cycle Counting with odd cycles. More concretely, we prove the following lemma.

Lemma 6.5.1. *Any single-pass streaming algorithm that distinguishes graphs of cycles with length $(2t+1)$ from cycles with length $2 \cdot (2t+1)$ with probability at least $\frac{99}{100}$ has to use a memory of $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits.*

To prove Lemma 6.5.1, we need to formally introduce the Boolean Hidden Hypermatching (BHH) problem as follows.

Definition 6.5.2 (Boolean Hidden Hypermatching ($\mathsf{BHH}_{m,t}$), [VY11,LW16,AKL17])**.** *The Boolean Hidden Hypermatching problem (denoted as $\mathsf{BHH}_{m,t}$) is a one-way communication problem between two players, namely Alice and Bob. Alice is given a boolean vector*

$x \in \{0,1\}^m$, where where $m = 2kt$ for some integer $k \geqslant 1$ and Bob gets a perfect t-hypermatching M on m vertices, and a boolean vector $w \in \{0,1\}^{\frac{m}{t}}$. Let Mx denote the length $\frac{m}{t}$ boolean vector $(\bigoplus_{1\leqslant i\leqslant t} x_{M_1,i}, \cdots, \bigoplus_{1\leqslant i\leqslant t} x_{M_{m/t},i})$, where $\{M_{1,1}, \cdots, M_{1,t}\}$, ..., $\{M_{m/t,1}, \cdots, M_{m/t,t}\}$ are the indices corresponds to the vertices matched by hypermatching $M = \{M_1, \cdots, M_{m/t}\}$. It is promised that either $Mx \bigoplus w = 0$ or $Mx \bigoplus w = 1$. The goal of the problem is to output Yes *when $Mx \bigoplus w = 0$ and* No *when $Mx \bigoplus w = 1$.*

Note that we use m to denote the number of vertices in the Boolean Hidden Hypermatching to avoid ambiguity in the reduction, where we are going to construct a new graph with n vertices. It is known that solving $\mathsf{BHH}_{m,t}$ is hard even when Bob's vector is deterministically all zero. More formally, the following result is standard in the literature.

Proposition 6.5.3. *(Communication Complexity of $\mathsf{BHH}_{m,t}$, [VY11]) Let $m = 2kt$ for some integer $k \geqslant 1$ and $t \geqslant 2$ as prescribed in Definition 6.5.2. Any one-way communication protocol that solves $\mathsf{BHH}_{m,t}$ with probability at least $\frac{99}{100}$ requires $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits in the message. Furthermore, the communication complexity holds even Bob's vector is known to be $\mathbf{0}^{m/t}$.*

We will use the hardness result of Proposition 6.5.3 to prove the hardness of a new version of the Gap Cycle Counting problem, defined as follows.

Definition 6.5.4 (Gap Cycle Counting – Odd Cycle)**.** *The Gap Cycle Counting with odd cycle problem is a one-way communication problem between two players, namely Alice and Bob. The input to the players is as follows.*

- *Both players are given n vertices, of which $\frac{1}{2t+1}$ fraction of the vertices are* special.
- *Alice is given a set of edges E_A which is a matching between the vertices that are* not *special.*
- *Bob is given a set of edges E_B which consists of two edges on each of the special vertex.*

The promise is that $E_A \cup E_B$ is either a disjoint union of cycles with $(2t+1)$ length, or a disjoint union of cycles with $2 \cdot (2t+1)$ length. The goal for the players is to answer Yes *for the former scenario and* No *for the latter.*

We show that the above GCC with odd cycle problem requires the same communication complexity as Definition 6.5.2.

Claim 6.5.5. *Any one-way communication protocol that solves the Gap Cycle Counting – Odd Cycle problem with probability at least* $\frac{99}{100}$ *requires* $\Omega\left(n^{1-\frac{1}{t}}\right)$ *bits in the message.*

Proof. We prove this by a reduction from the $\mathsf{BHH}_{m,t}$ problem. Suppose we have a communication protocol PROT that solves Gap Cycle Counting – Odd Cycle problem with probability at least $\frac{99}{100}$ and a message of $o\left(n^{1-\frac{1}{t}}\right)$ bits, we show that it is possible to get a communication protocol that solves $\mathsf{BHH}_{m,t}$ with probability $\frac{99}{100}$ and the same message. This contradicts the lower bound of Proposition 6.5.3. Therefore, the only conclusion is that such a PROT cannot exist.

The communication protocol to solve $\mathsf{BHH}_{m,t}$ can be given as follows.

PROT': a communication protocol for $\mathsf{BHH}_{m,t}$
Input: An instance of $\mathsf{BHH}_{m,t}$.
Output: The decision of whether $Mx = 0$ or $Mx = 1$.

1. Alice and Bob both create an all-one vector $y = \mathbf{1}^{2m/t}$, and append it after their own vectors (without communication).
2. Alice locally creates vertices and edges:
   (a) For each x_i, create 4 vertices: u_i, v_i, u'_i, v'_i.
   (b) If $x_i = 0$, add $(u_i, v_i) \in E_A$ and $(u'_i, v'_i) \in E_A$.
   (c) If $x_i = 1$, add $(u_i, v'_i) \in E_A$ and $(u'_i, v_i) \in E_A$.
   (d) For each y_j, create two vertices $\tilde{u}_j, \tilde{u}'_j$, and add no edge to them.
3. Bob locally creates vertices and edges:
   (a) For each vertex z_i in the $\mathsf{BHH}_{m,t}$ instance, create vertices u_i, v_i, u'_i, v'_i in the same manner of Alice.
   (b) For each hyperedge M_j, denote $(M_{j,1}, M_{j,2}, \cdots, M_{j,t})$ as the indices of the vertices.
   (c) For $\ell = 1$ to $t-1$, Bob adds edges between $(v_\ell, u_{\ell+1}) \in E_B$ and $(v'_\ell, u'_{\ell+1}) \in E_B$.
   (d) Bob further create two vertices $\tilde{u}_j, \tilde{u}'_j$ for each y_j, and assign to j-th matching by adding four edges: $(v_t, \tilde{u}_j) \in E_B$, $(v'_t, \tilde{u}'_j) \in E_B$, $(\tilde{u}_j, u_1) \in E_B$, $(\tilde{u}'_j, u'_1) \in E_B$.

4. The players runs PROT, and answers as the output of PROT, i.e. answer Yes if PROT returns Yes (graph is of $(2t+1)$ cycles), and vice versa.

An intuitive understanding of the edges added by Alice can be found in Figure 6.1. Observe that as in the figure, if we always put vertices u and v to the 'outer' circle of the vertices and u' and v' to the 'inner' one, then the edges added by Alice are 'parallel' when $x_i = 0$ and 'crossing' when $x_i = 1$. Bob always adds 'parallel' edges between the corresponding vertices.

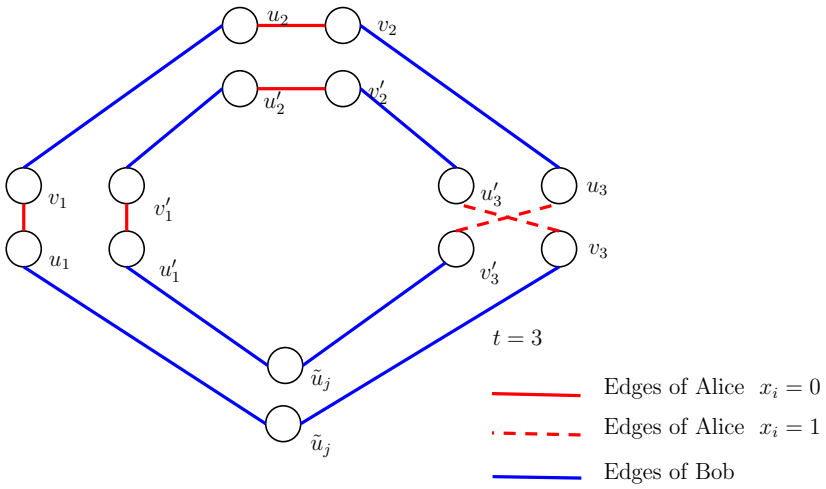


Figure 6.1: The $\mathsf{BHH}_{m,t}$ instance to Gap Cycle Counting with odd cycles. In the instance of the figure, Alice has $x_3 = 1$ (for matching j). Therefore, we have $Mx = 1$ and the graph is one 14-length cycle. One can verify that if one more set of 'cross edges' is added, the graph goes to two separate cycles.

It is straightforward to verify that the inputs constructed by Alice and Bob is valid for PROT: the fraction of the special vertices is indeed $\frac{2m/t}{4m+2m/t} = \frac{1}{2t+1}$. We now show that the graph constructed by the two players is with cycles of length $(2t+1)$ if and only if $Mx = 0$, and cycles of length $2 \cdot (2t+1)$ if and only if $Mx = 0$. Note that for a graph with only 'parallel' edges, each matching in the $\mathsf{BHH}_{m,t}$ instance induces two cycles of length $(2t+1)$. If one of the vertex pairs switches to 'cross' edges, the tours of the cycles intersect and the two cycles merge into one cycle with length $2 \cdot (2t+1)$. Inductively, one can show that if there is an even number of 'cross' edge connections, the two $(2t+1)$-length cycles remain separate; and if there is an odd number of 'cross' edge connections, the two $(2t+1)$-length cycles merge. Therefore, the type of cycle induced by the $\mathsf{BHH}_{m,t}$ instance depends

uniquely on the value of Mx for each matching. Furthermore, the cycles are constructed independently from each matching, and there are no other edges. This gives us the desired correspondence between the length of the cycles and the value of Mx.

Since the assumption is that PROT succeed with probability at least $\frac{99}{100}$, and the reduction in PROT' is *deterministic*, PROT' can answer $\mathsf{BHH}_{m,t}$ successfully with the same probability. Therefore, any such PROT must have a communication complexity of at least $\Omega\left(n^{1-\frac{1}{t}}\right)$. ∎

Proof of Lemma 6.5.1. The proof of Lemma 6.5.1 now follows from a standard reduction by Claim 6.5.5. Suppose a streaming algorithm that can distinguish graphs of cycles $2t+1$ from graphs of cycles $2 \cdot (2t+1)$ exists, one can use the streaming algorithm to simulate a communication protocol: Alice runs the streaming algorithm, send the memory of the streaming algorithm as the message, and Bob uses the memory to run the rest of the streaming algorithm. Therefore, such a streaming algorithm must use a memory of at least $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits.

6.5.2 The lower bound

We now formally state and prove the additive approximation gap for correlation clustering. To start with, we first show a lower bound for *dynamic* streaming algorithms that allows the operations of the insertion and removal of $(+)$ and $(-)$ edges (see Section 6.2.3 for the detailed definition). Switching to dynamic streams simplifies the reduction, and we will deal with an additional technical part to prove the lower bound for insertion-only streaming algorithms later.

Lemma 6.5.6. *Let $c \in [1, \frac{6}{5})$ and $\varepsilon \in (0, \frac{6}{5} - c)$. Any single-pass* dynamic *streaming algorithm that estimates the optimal value OPT of correlation clustering by a $(c, \varepsilon n)$-approximation (i.e., an estimated value that is at most $(c \cdot \mathsf{OPT} + \varepsilon \cdot n)$) with probability at least $\frac{99}{100}$ has to use a memory of $\Omega(\sqrt{n})$ bits, even on labeled (complete) graphs.*

Proof of Lemma 6.5.6. We use the simple instance of cycles with length 5 versus cycles of length 10 from the results of Lemma 6.5.1. We again use the reduction argument: given a dynamic streaming algorithm ALG that achieves a (c, ε)-approximation for the optimal cost of correlation clustering on labeled graphs, we can design a streaming algorithm to distinguish graphs with cycles of length 5 and cycles of length 10 in the following way:

1. (−) edges: By the start of the stream, for each vertex pair (u, v), insert a (−) edge.

2. (+) edges: For each arriving edge $(u, v) \in E$, remove the (−) edge between (u, v), and add a (+) edge.

3. Run ALG and output with the following rules: if the cost is less than $\frac{3n}{5}$, output *No* (graph of cycles of length 10); otherwise, output *Yes* (graph of cycles of length 5).

We show that provided the correct probability of ALG, we can distinguish the scenarios of length-5 and length-10 cycles with the same probability. Note that by the construction, the (+) edges are the cycle edges. For a graph with (+) edges of length-10 cycles, there is a way to induce a cost of $\frac{n}{2}$: on each cycle, we can put two vertices in the same cluster, and create 5 clusters. In this way, we pay a cost of 5 by splitting (+) edges, and there are $\frac{n}{10}$ such cycles.

On the other hand, when the graph is of disjoint cycles of length 5, we show that the cost is at least $\frac{3n}{5}$. To see this, we first note that an optimal clustering never puts two vertices from different cycles in the same cluster, as there are only (−) edges between them. Among any cycle, the optimal clustering is to put 2 vertices in one cluster, and the other 3 in the same cluster. This induces a cost of 3, and there are $\frac{n}{5}$ such cycles, which gives us the desired solution.

Finally, note that if ALG achieves (c, ε)-approximation in the given parameter range, the value is strictly less than $\frac{3n}{5}$ when the graph is of disjoint length-10 cycles. Therefore, we get the desired reduction. ∎

We now generalize the results to insertion-only streams. To this end, we will need the following variation of the Gap Cycle Counting with odd cycles.

Lemma 6.5.7. *Suppose a complete graph has two types of edges: the* normal *edges and the* special *edges. Let G be a graph whose* special *edges form disjoint cycles of length $(2t+1)$, and $\tilde{G}$ be a graph whose* special *edges forms disjoint cycles of length $2 \cdot (2t+1)$. Any single-pass streaming algorithm that distinguishes G from $\tilde{G}$ with probability at least $\frac{99}{100}$ has to use a memory of $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits.*

Proof. We can prove Lemma 6.5.7 by slightly tweaking the Gap Cycle Counting with odd cycle problem of Definition 6.5.4. Now suppose the edges E_A and E_B that are given to

Alice and Bob are the *special* edges, and we further provide other *normal* edges to make the graph complete. The goal for the players is to distinguish the length of the cycles from the *special* edges.

Let us call this problem GCC-odd-C ('C' stands for 'complete'). There is a straightforward reduction from the $\mathsf{BHH}_{m,t}$ problem to this new variate GCC-odd-C. Suppose we have a communication protocol that solves GCC-odd-C. Then, Alice can mark all the edges she added in the reduction of Claim 6.5.5 as *special*, and add *normal* edges between all other pairs between (u_i, v_i, u'_i, v'_i) for each i. Bob can similarly mark all the edges he added in the reduction of Claim 6.5.5 as *special*, and add *normal* edges between the vertex pairs of different ℓ and $(\tilde{u}_j, \tilde{u}'_j)$ whenever there is not a special edge. Such a protocol can solve $\mathsf{BHH}_{m,t}$ with output of GCC-odd-C. Therefore, any communication protocol that solves GCC-odd-C requires $\Omega\left(n^{1-\frac{1}{t}}\right)$ bits communication, and the streaming lower bound follows. ∎

Finalizing the proof of Theorem 12. We can prove the lower bound for insertion-only streaming algorithms based on Lemma 6.5.7. Note that now we can mark each *special* edge as a $(+)$ edge, and each *normal* edge as a $(-)$. The graph in GCC-odd-C is complete, which means we can construct an input to the insertion-only streaming algorithm without any extra memory cost. As such, combining the proof of Lemma 6.5.6 and the result of Lemma 6.5.7 yields the result.

Part II

Part II: Modern Models of Computation

Chapter 7

Fully Dynamic 4-Cycle Counting

7.1 Introduction

We study subgraph counting over fully dynamic graphs that undergo edge insertions and deletions. Given an input graph G and a small pattern graph H, say a triangle or a 4-cycle, we are interested in counting the number of subgraphs (not necessarily induced) of G that are isomorphic to H. Counting subgraphs is a fundamental problem in graph theory with numerous applications across various fields, including social network analysis, telecommunication networks, and computational biology (see, e.g., [RPS+21, COJT+11, MSOI+02, SMS+20] for some applications).

Subgraph counting has many applications in database theory [NRR14, NPRR18, KNN+20] as well. The problem of counting the number of elements in binary joins over binary relations is equivalent to the problem of subgraph counting in **layered graphs** (formally defined in Section 7.2), where the vertex set is partitioned into multiple layers, each corresponding to an attribute. Vertices represent attribute values, and edges exist only between specific layers (typically consecutive ones), representing tuples of attribute values in the relations.

Specifically, the problem of finding the number of elements in a cyclic join of size k is equivalent to counting the number of k-cycles in k-layered graphs. For example, let R, S, and T be relations that have schemas (A, B), (B, C), and (C, A) respectively. Then the size of the join of R with S with T equals the number of triangles in the corresponding layered graph where there is a layer for each attribute A, B, C, the vertices are the attribute values and the edges only exist between neighboring layers representing the tuples of attribute

values in the relations. Throughout this paper, we adopt the language of subgraph counting; however, all results can equivalently be rephrased in terms of joins. An example of the connection between joins and layered graphs can be found in Figure 7.1.

In the dynamic model, the pattern graph H is fixed and known apriori, and the input graph G starts as an empty graph which then undergoes edge insertion and deletions. If both insertions and deletions are allowed, the model is referred to as fully dynamic. After *every* update we need to output the *exact* number of occurrences of the subgraph H inside G. The goal is to minimize the worst-case update time over all edge updates.

In this paper, we are interested in counting 4-cycles in the fully dynamic model. We can equivalently state the problem as follows: Given four binary relations $A(L_1, L_2)$, $B(L_2, L_3)$, $C(L_3, L_4)$, and $D(L_4, L_1)$ that undergo continuous updates through tuple insertions and deletions, the problem is to dynamically maintain the count of tuples in the cyclic join $J = A \bowtie B \bowtie C \bowtie D$. While J itself is not materialized, a view storing its count is maintained and must be updated after every update to the database. As updates occur, incremental view maintenance (IVM) is used to efficiently update the join size by computing the change in count (ΔJ) rather than recomputing the join size from scratch. The goal is to minimize the worst-case update time over all updates to the database while ensuring that the updated count is output after each update.

We are interested in 4-cycles because it is the smallest pattern graph for which the dynamic counting problem is not well-understood. Counting subgraphs on 1 and 2 vertices is trivial and counting subgraphs on 3 vertices is well studied. In particular, 3-cycle (triangle) counting in the dynamic model has an upper bound of $O(\sqrt{m})$ for the worst-case update time [KNN+20]. There is also a conditional lower bound of $\Omega(m^{1/2-\gamma})$ for any constant $\gamma > 0$ for the update time [HKNS15] under the Online Matrix-Vector (OMv) conjecture.

However, the bounds for graphs on 4 vertices are not well understood. In particular, 4-cycle counting in the dynamic model has an upper bound of $O(m^{2/3})$ [HHH22] for the worst-case update time but the conditional lower bound is the same bound of $\Omega(m^{1/2-\gamma})$ for any constant $\gamma > 0$ [HKNS15, HHH22]. [HHH22] also studied the problem of counting 4-cliques showing that the folklore upper bound of $O(m)$ for the update time is tight under the static combinatorial 4-clique conjecture. One may imagine that the complexity of the update time for 4-cycles should be somewhere between $O(\sqrt{m})$ for triangles and $O(m)$ for 4-cliques. This raises the following question:

> Question: What is the worst-case update time for counting 4-cycles in the fully dynamic model?

The algorithm of [HHH22] is a natural generalization of the approach for counting triangles in [KNN+20], so it seems natural that $O(m^{2/3})$ (which sits in the middle of $O(\sqrt{m})$ and $O(m)$) might be the correct answer for the update time for counting 4-cycles. We show this is *not* the case by giving an algorithm for maintaining the number of 4-cycles in fully dynamic graphs in $O(m^{2/3-\varepsilon})$ worst-case update time for some constant $\varepsilon > 0$. This shows that $O(m^{2/3})$ is not the correct answer for the complexity of the update time. Consider the following theorem:

Theorem 13. *There is an algorithm for counting the number of 4-cycles in any fully dynamic graph in worst-case update time $O(m^{2/3-\varepsilon})$ where m is the number of edges in the graph at that point and $\varepsilon > 0$ is a constant. Using the current best value of the square matrix multiplication exponent $\omega = 2.371339$ [1] we get $\varepsilon = 0.009811$ and if we assume the best possible exponent i.e. $\omega = 2$ then we get $\varepsilon = 1/24$.*

This approach crucially relies on fast matrix multiplication, and we only get an improvement when $\omega < 2.5$. This is very surprising because any upper bound on ω better than 3 like Strassen's algorithm [Str69] is not sufficient. For $\omega = 2.371339$ the update time improves from $m^{0.66}$ to $m^{0.65686}$ and if $\omega = 2$ it further improves to $m^{0.625}$. This improvement to the second digit after the decimal place is significant considering improvements of a similar kind are much smaller like in metric TSP [KKG21], random-order streaming matching [AB21], and fast matrix multiplication [ADW+25]. These results, like ours, demonstrate that the previous best bounds were not optimal by providing a slight improvement over them.

Existing Lower Bounds. The lower bound of $\Omega(m^{1/2-\gamma})$ for any constant $\gamma > 0$ for the update time for maintaining the number of triangles in any fully dynamic graph holds under the OMv conjecture, so it even holds against algorithms that use fast matrix multiplication. However, the lower bound of $\Omega(m^{1-\gamma})$ for any constant $\gamma > 0$ for the update time for maintaining the number of 4-cliques in any fully dynamic graph holds under the static combinatorial 4-clique conjecture, so it does not hold against algorithms that use fast matrix multiplication. Thus, it may be possible to develop a faster dynamic algorithm for 4-cliques using fast matrix multiplication without breaking the lower bound. We note that our algorithm is non-combinatorial because it uses fast matrix multiplication, so we do not rule out a combinatorial lower bound of $\Omega(m^{2/3})$ for the update time for maintaining the number of 4-cycles in a fully dynamic graph.

We will show in Section 7.8 that the problem of dynamically maintaining the number of 4-cycles in general graphs is equivalent to dynamically maintaining the number of layered

[1] The time complexity of multiplying two $n \times n$ matrices is $O(n^{\omega})$.

4-cycles in 4-layered graphs (which we define formally in Section 7.2 and informally below). 4-layered graphs are graphs with 4 sets of vertices which we call layers each of which are independent sets and the edges of the graph only exist between consecutive layers (there can also be edges between the last and first layer). These graphs are basically 4-partite graphs with additional constraints on the edges (that edges exist only between consecutive layers). A layered 4-cycle in a 4-layered graph is a 4-cycle that has exactly one vertex in each layer and edges connecting these vertices in adjacent layers.

As we discussed earlier, counting layered cycles in layered graphs is interesting from the perspective of database theory. Suppose we have attributes L_1, L_2, L_3 and L_4 and binary relations $A(L_1, L_2)$, $B(L_2, L_3)$, $C(L_3, L_4)$, and $D(L_4, L_1)$ defined on these attributes. We can construct a 4-layered graph G to represent this information, where each layer corresponds to one of the attributes L_1, L_2, L_3, and L_4. Each element of the attributes is represented as a vertex in the graph G. Also, each tuple in the relations corresponds to an edge in G. If we compute the join of A, B, C and D we are effectively finding tuples (a, b, c, d) such that: $(a, b) \in A$, $(b, c) \in B$, $(c, d) \in C$, and $(d, a) \in D$. Each resulting tuple (a, b, c, d) from the join corresponds to a unique 4-cycle in the graph G. Therefore, the size of the join is equal to the number of layered 4-cycles in G.

Thus, the problem of computing the size of a join of four binary relations is equivalent to the problem of counting the number of layered 4-cycles in 4-layered graphs. Since maintaining the number of 4-cycles in a fully dynamic graph is equivalent to maintaining the number of layered 4-cycles in fully dynamic 4-layered graphs, we solve the latter problem. We state the result for 4-layered graphs below:

Theorem 14. *There is an algorithm for counting the number of layered 4-cycles in any fully dynamic 4-layered graph in worst-case update time $O(m^{2/3-\varepsilon})$ where m is the number of edges in the graph at that point and $\varepsilon > 0$ is a constant. Using the current best value of the square matrix multiplication exponent $\omega = 2.371339$ we get $\varepsilon = 0.009811$ and if we assume the best possible exponent i.e. $\omega = 2$ then we get $\varepsilon = 1/24$.*

This is the first result to demonstrate that fast matrix multiplication can be leveraged to achieve a speedup in join size estimation problems in dynamic graphs. While this approach has been previously explored for static algorithms [T̆22, Hu24], it had not been extended to the dynamic setting until now. For example, in the static case, fast matrix multiplication enables solving the triangle query in time $O(m^{\frac{2\omega}{\omega+1}})$ [AYZ97], which is faster than the traditional combinatorial $m^{3/2}$ algorithm since $\omega < 3$. However, in dynamic settings, such a gap between combinatorial and non-combinatorial algorithms had not been observed. The triangle query, for instance, can be maintained in $O(m^{1/2})$ update time using

a combinatorial algorithm, which is optimal under the OMv conjecture. A widely held belief was that fast matrix multiplication does not help in dynamic settings, as algorithms like Strassen's require prior knowledge of the two input matrices. Consequently, it was assumed that Incremental View Maintenance (IVM) would not benefit from fast matrix multiplication. Our result challenges that intuition by providing a counterexample using the 4-cycle query. This insight could potentially inspire a new class of IVM algorithms.

It is also important to note that we get a worst-case bound on the update-time and not an amortized bound. On the surface it looks like we are multiplying large matrices, so the bound should be amortized, but we answer initial queries quickly because there are only a few edges and in parallel compute the matrix products using fast matrix multiplication to speed up later queries. We now discuss our techniques in more detail.

Our Techniques

To solve the problem of maintaining the total number of 4-cycles we count the number of cycles through the new update. Thus, the algorithm remains the same (counts the number of cycles through the new edge update) whether the update is an insertion or deletion. The answer for the total number of 4-cycles after the update is the total number of 4-cycles before the update plus or minus the number of cycles through the new edge update depending on whether it is an insertion or deletion. We count the number of cycles through the new edge update by finding the number of 3-paths between the endpoints of the edge update. In the simple algorithm (Section C.1), we do this by storing a data structure for the number of 2-paths (paths with 2 edges also called *wedges*) between all pairs of vertices. During a query (u, v), we count the number of 3-paths between u, v by going over all neighbors of u and then using the stored number of wedges to v and then summing them up to get the number of 3-paths. This gives a worst-case update time of $O(n)$. We now want to get the update time purely as a function of m, the number of edges. The goal is to minimize the worst-case update time over all edge updates. This might be slightly tricky because we want the update time to be purely a function of m, the current number of edges in the graph, but m changes with every insertion or deletion. Thus, the goal is to come up with an algorithm with a worst-case update time of $O(m^x)$ with the smallest possible value of x.

Algorithm of Previous Work [HHH22] is successful in getting the update time purely as a function of m and give an upper bound of $O(m^{2/3})$ for the worst-case update time. Their approach is to group vertices of the graph into high and low degree vertices and compute different data structures for them which can be maintained in $O(m^{2/3})$ update time and use them to answer queries in $O(m^{2/3})$ update time. The data structures they

maintain for low degree vertices are wedges through low degree vertices and 3-paths through two low degree vertices, both of which can be maintained because iterating over pairs of neighbors of low degree vertices can be done easily. For the high degree vertices, they maintain wedges through high degree vertices but only for pairs of vertices that have a high degree. Maintaining this is easy because the number of high degree vertices is small. For the query, the number of paths through two low degree vertices is stored. The number of paths through one high and one low degree vertex can be found easily by iterating over all high degree vertices and using the wedges through low degree vertices. To calculate the number of paths through two high degree vertices, they do different things depending on the degrees of the endpoints of the query. If the endpoints have a low degree then they iterate over their neighbors otherwise they use the wedges through high degree vertices. This is enough to give them an update time of $O(m^{2/3})$.

Our Algorithm To improve the upper bound of [HHH22] we crucially use fast matrix multiplication. We group the vertices into many classes based on their degree and the layer they are in and create data structures for them and use them to answer queries in update time $O(m^{2/3-\varepsilon})$. The main technical part is to solve the problem under the assumption that the number of vertices in each layer $n = m^{2/3}$ (this is technically incorrect because we assume $n = m^{2/3+2\varepsilon}$ but this is morally correct). Dropping this assumption is highly non-trivial, but we show how to do that in Section 7.6. The idea is that for vertices that have a very small degree, we can answer queries for them and compute paths through them in $O(m^{2/3-\varepsilon})$ update time, so we can ignore them for the rest of the analysis. This gives us a lower bound on the degree of the remaining vertices and hence we can assume that the number of relevant vertices is small ($m^{2/3}$).

The vertices in the first and last layer are divided into high, medium, and low degree. The vertices in the second and third layer are divided into high and low degree called dense and sparse vertices respectively. We can iterate over the neighbors of low and medium degree vertices within the update time, and we can iterate over all the medium and high degree vertices within the update time. The first data structure we store is wedges through sparse vertices (similar to wedges through low degree vertices). Next we store wedges through dense vertices but the endpoints of the wedges have to be either high, medium or dense (similar to wedges through high degree vertices, but the endpoints are also high degree). Using these we can answer queries when one of the endpoints is medium degree which without loss of generality is in the first layer. We can iterate over the neighbors of the vertex in the first layer and then use the number of wedges through sparse vertices to the other endpoint. To get the paths through the dense vertices in the third layer, we iterate over all of them and check if they have an edge to the endpoint in the last layer. We then use the wedges through sparse vertices and wedges through dense vertices to the

endpoint in the first layer. These details are similar to the algorithm of [HHH22].

The first difficult case is when both the endpoints of the query are high degree, and we have to calculate the number of paths through two sparse vertices. The algorithm of [HHH22] had the number of 3-paths through two low degree vertices stored, but we cannot afford to store the number of 3-paths through two sparse vertices. This is where our crucial idea of **phases** comes in. We partition the stream of updates into phases of size $m^{1-\delta}$, and it is important that the size of a phase is asymptotically smaller than m, so we need $\delta > 0$. We then consider eight different types of 3-paths where each edge of the 3-path is either in the current phase or in an old phase. For most types of these 3-paths, we are able to store the number of 3-paths through two sparse vertices like the data structure for the number of 3-paths through two low degree vertices in [HHH22]. This is because the number of high degree vertices when considering the edges of a single phase is asymptotically smaller compared to the number of high degree vertices when considering all the edges. Using these ideas (and some others) we are able to answer queries when both endpoints of the query have a high degree.

Another difficult case is when both the endpoints of the query are low degree, and we have to calculate the number of paths through two dense vertices. We cannot iterate over the neighbors of both endpoints like we did in the algorithm of [HHH22]. We need to use another property of phases here. A phase should be long enough so that in the time it takes to process all the edge updates in a phase, we are able to multiply two square matrices of dimension $m^{2/3}$. During a phase, we compute the number of paths between all pairs of vertices in the previous phase using fast matrix multiplication. This allows us to iterate over the neighbors of an endpoint and use the stored number of wedges to the other endpoint that go through the edges of the old phases. To deal with edges that go through the new phase we take advantage of its size and iterate over all pairs of dense vertices in the second and third layer (one of which has just $O(m^{1-\delta})$ edges incident on it) and compute the number of paths through them. A combination of these ideas along with some others allows us to answer queries when both endpoints have a low degree.

Note that our algorithm is not as simple as running the algorithm of [HHH22] for the new phase (this would give a better runtime because we have asymptotically fewer edges) and then using matrix multiplication to calculate the number of paths between all pairs of vertices in the old phases. This is because we also need to look at paths that have some edges in the old phases and some in the new phase. We spell out the full details of the algorithm in Section 7.5 with the setup being in Section 7.4. We also state a warm up to the main algorithm in Section 7.3 which solves the problem under an assumption. The purpose of stating this algorithm is two-fold. The algorithm is simpler so this gives a flavor for the main algorithm, and we will also use it as a subroutine in our main algorithm.

Related Work. We now discuss relevant related work. [ES09] studied the problem of counting the number of 3-vertex subgraphs in a fully dynamic graph and gave an algorithm with update time $O(h)$ where h is the h-index of the graph, the maximum number such that the graph contains h vertices of degree at least h. [EGST12] extend the results of [ES09] to directed subgraphs and to subgraphs of size 4. For the directed case, they provide an algorithm for all three-vertex induced subgraphs in $O(h)$ amortized update time. For the undirected case, they provide an algorithm for maintaining the counts of 4-vertex subgraphs in $O(h^2)$ amortized update time. [HLS22] showed that counting 4-cycles is hard even in random graphs by giving a lower bound of $\Omega(m^{1/2-\gamma})$ for any $\gamma > 0$, for the update time. Finally, [HHS22] provides a comprehensive survey on recent advances in fully dynamic graph algorithms.

Outside maintaining the join size in dynamic graphs, there has also been related work on join algorithms [NRR14, NPRR18], join size estimation and join sampling [DLT23, KHFH23] among other areas. There is also a vast body of work on cycle detection and approximately counting the number of cycles for both directed and undirected graphs starting with [AYZ97, YZ97] and and culminating in the current best bounds [T$\check{\text{a}}$22, CHEVW24] (see additional references therein). The problem of approximately counting subgraphs, particularly triangles and 4-cycles, has attracted significant attention in the streaming setting [JG05, BFL$^+$06, BBCG10, KMSS12, MVV16, KMPV19, MV20, Vor20]. Finally, there is also a long line of work for listing cycles beginning with the classic algorithms [Tar73, Joh75] with recent advancements focusing on efficiently listing 6-cycles in both dense and sparse graphs [JWZ24, VWW25].

7.2 Preliminaries

Note that in this paper, we will only consider unweighted undirected simple graphs (no multi-edges and self loops).

A **k-path** in a graph is a path with k-edges. Note that we only consider simple paths, so every vertex on the path has degree exactly 2 except for the starting and ending vertex which have degree 1. Formally, a k-path in G is a sequence of distinct vertices $(v_0, v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leqslant i < k$. In this paper, we will mostly just use 2-paths (also called wedges) and 3-paths.

A **k-cycle** in a graph is a cycle with k-edges (equivalently k vertices). We only consider simple cycles, so every vertex in the cycle has degree exactly 2. Formally, a k-cycle in G is a sequence of vertices $(v_0, v_1, \ldots, v_{k-1}, v_0)$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leqslant i < k$ (with indices taken modulo k), and where all vertices $v_0, v_1, \ldots, v_{k-1}$ are distinct. In this paper, we will primarily focus on 4-cycles.

Databases	
A (L_1, L_2)	B (L_2, L_3)
(1,1)	(1,1)
(1,2)	(2,1)
(1,3)	(3,1)
(2,2)	(3,3)
(3,2)	

$A \bowtie B$
(1,1,1)
(1,2,1)
(1,3,1)
(1,3,3)
(2,2,1)
(3,2,1)

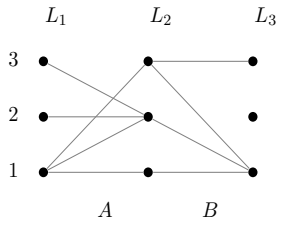


Figure 7.1: Binary Relations A, B. Every element of the join $(A \bowtie B)$ corresponds to a 2 -paths in the layered graph between L_1 and L_3. Thus, the size of the join $(A \bowtie B)$ is the number of 2-paths in the layered graph.

A **4-layered graph** is a graph $G = (V, E)$ where $V = L_1 \cup L_2 \cup L_3 \cup L_4$ is a union of four layers each containing n vertices. Each layer is an independent set (no edges exist between vertices in the same layer). Additionally, edges only exist between the four consecutive layers and are represented by the matrices A, B, C, and D.

A **layered 4-cycle** in a 4-layered graph $G = ((L_1, L_2, L_3, L_4), E)$ is a 4-cycle defined by a sequence of vertices $(v_1, v_2, v_3, v_4, v_1)$ such that $v_i \in L_i$ and $(v_i, v_{i+1}) \in E$ for all $i \in [4]$ (where $v_5 = v_1$).

We can similarly define layered 2-paths and layered 3-paths which have all of its vertices in different layers. Throughout the paper, we are solely focused on layered cycles and paths in 4-layered graphs, so for convenience, we will drop the word "layered".

Fast Matrix Multiplication. We now discuss some notation for fast matrix multiplication. Note that ω is the exponent of square matrix multiplication. This means that the time it takes to multiply two $n \times n$ matrices is $O(n^\omega)$. The current best known upper bound on ω is 2.371339 [ADW+25]. We will also need fast rectangular matrix multiplication in our algorithm. We define $\omega(a, b, c)$ to be the exponent of rectangular matrix multiplication of an $n^a \times n^b$ dimension matrix by an $n^b \times n^c$ dimension matrix. [ADW+25] also gives improved upper bounds for $\omega(a, b, c)$.

Equivalent Queries. After every update to the simple graph, the query we have to answer is the exact total number (count) of distinct 4-cycles in the current graph. We will show in Section 7.8 that this problem is equivalent in general and layered graphs, so we will just focus on layered graphs. Thus, the new problem is the following. After every update to the simple 4-layered graph, the query we have to answer is the exact total number of distinct layered 4-cycles in the current graph. This is the same problem as having 4 binary

relations that are continuously updated, where the query answered after every update is the exact total number of tuples in their cyclic join.

It is easy to see that maintaining the total number of layered 4-cycles is equivalent to just counting the number of new layered 4-cycles i.e. cycles that go through the new edge update (insertion or deletion). This is because we can add or subtract cycles that go through the new edge update from the previous answer to get the new answer. We answer the query of the number of new layered 4-cycles by counting the number of 3-paths between the endpoints of the edge update.

Without loss of generality, we assume that queries have to be answered only when there is an edge update in D because we can run 4 copies of this algorithm, one for queries in D and the others for queries in A, B, and C. After each update, we ask the relevant instance of the algorithm to provide the answer. This means we now focus on queries in D while handling edge updates in A, B, and C. To efficiently answer queries in D, we maintain appropriate data structures for edges in A, B and C. Note that we maintain the same data structures for all 4 copies of the algorithm (corresponding to matrices A, B, C and D). By symmetry, the analysis is the same for all of them, so we just focus on the case when queries are in D and edge updates are everywhere else. The final equivalent problem we will solve is the following:

> We are given a 4-layered graph G which undergoes edge updates in A, B and C each of which must be processed in $O(m^{2/3-\varepsilon})$ worst-case time. At any point, a query (u, v) can be made ($u \in L_1$ and $v \in L_4$) for the number of 3-paths between u and v that go through A, B and C which should be answered in $O(m^{2/3-\varepsilon})$ worst-case time.

7.3 Warm-up: When A and C are Fixed

In this section, we give an algorithm for counting 4-cycles in a 4-layered graph under a few assumptions.

7.3.1 Setup

We let the worst-case update time be $O(m^{2/3-\varepsilon_1})$ and we will show we can get $\varepsilon_1 > 0$ to be a constant. We first discuss the data structures we need to store and then show how the queries can be answered in time $O(m^{2/3-\varepsilon_1})$.

Before we discuss the data structures, we group the vertices into different **classes**. The vertices in L_1 (resp. L_4) are partitioned into H, M, L based on their degree in A (resp. C).

- **High** (H): degree between $m^{2/3-\varepsilon_1}$ and n.
- **Medium** (M): degree between $m^{1/3+\varepsilon_1}$ and $2m^{2/3-\varepsilon_1}$.
- **Low** (L): degree between 0 and $2m^{1/3+\varepsilon_1}$.

The edge updates to B are divided into **chunks** of size $m^{2/3-\varepsilon_1}$ called $B_1, B_2, \ldots B_i$. For a chunk B_i, vertices in L_2 and L_3 with degree in B_i at most $m^{1/3-\varepsilon_2}$ are in **Sparse** (S) and degree in B_i at least $m^{1/3-\varepsilon_2}$ are in **Dense** (D). Note that these classes are different for each B_i, so a vertex could be in S for some chunk B_i and in D for some other chunk B_j. We also partition the edges of B_i based on whether the endpoints are in D or S. Thus, B_i can be written as a sum of four matrices: $B_i = B_{i,DD} + B_{i,SS} + B_{i,DS} + B_{i,SD}$. Also, we denote by $B_{<i} := \sum_{j=1}^{i-1} B_j$. Similarly, we have $B_{<i,DD} := \sum_{j=1}^{i-1} B_{j,DD}$ for all the four subsets of B_i. Note that these matrices may contain numerous rows and columns consisting entirely of zeros, effectively reducing their dimension for computational purposes.

We use superscripts to talk about a submatrix of a matrix by restricting the vertices in different layers. We also use $*$ to represent all the vertices in a layer. For instance A^{H*} represents the submatrix of A where we only look at the edges of A between L_1^H and L_2. We use a subscript to represent a subset of the edges. For instance B_i contains only the edges in chunk i and $B_{i,DD}$ only contains the edges between dense vertices of B_i. To make things less cumbersome we will drop some parts of the notation when things are clear from context. We now state the assumptions we need in this section:

Assumption 3. *The number of vertices in each layer is $n \leqslant m^{2/3+2\varepsilon}$.*

Assumption 4. *The vertices do not change classes throughout the algorithm.*

Assumption 5. *There are no edge updates in A and C i.e. the only edge updates are in B, D.*

We will prove the following lemma in this section:

Lemma 7.3.1. *Theorem 14 holds under Assumption 3, Assumption 4, and Assumption 5.*

High Level Idea. We would like to store wedges between all pairs of vertices like in the simple algorithm in Section C.1 and then during a query, iterate over the all the neighbors of one of the query vertices and use the number of wedges to the other query vertex. There are two major problems with this approach. We do not have enough time to iterate over

all the neighbors of a high degree vertex. Also, iterating over all low degree vertices in a layer takes too much time, and so we cannot store the wedges for low degree vertices.

Thus, we create different data structures for vertices in H, M, L which we show in the next subsection (Section 7.3.2). We directly store the number of 3-paths between pairs of high degree vertices because we cannot iterate over their neighbors. We store the wedges for high and medium degree vertices and some types of wedges for the low degree vertices. Lastly, we show how we use these data structures to answer queries in the subsection after we show the data structures (Section 7.3.3).

7.3.2 Data Structures

Data Structures		
High	**Medium**	**Low**
$A^{H*} \cdot B_{<i}$ (Eq (7.1))	$A^{M*} \cdot B_{<i}$ (Eq (7.3))	$A^{L*} \cdot B_{<i,DD}$ (Eq (7.4))
$A^{H*} \cdot B_{<i} \cdot C^{*H}$ (Eq (7.1))	$B_{<i} \cdot C^{*M}$ (Eq (7.3))	$A^{L*} \cdot B_{<i,S*}$ (Eq (7.4))
$B_{<i} \cdot C^{*H}$ (Eq (7.1))		$B_{<i,DD} \cdot C^{*L}$ (Eq (7.4))
		$B_{<i,*S} \cdot C^{*L}$ (Eq (7.4))

Table 7.1: Data Structures used in the Algorithm where A and C are fixed.

We first show what data structures we maintain. We maintain some data structures on the fly as the updates arrive and show that the update time is $O(m^{2/3-\varepsilon_1})$. For the remaining data structures, we compute them for the previous chunk when the updates of the next chunk arrive. We show that this process takes $O(m^{4/3-2\varepsilon_1})$ total time, so the computation can be spread over the $m^{2/3-\varepsilon_1}$ updates that arrive during the chunk, leading to our desired worst-case update time. The list of data structures can be found in Table 7.1.

Lemma 7.3.2. *The worst-case update time for all the data structures we store is $O(m^{2/3-\varepsilon_1})$ during an edge update and $O(m^{4/3-2\varepsilon_1})$ if it is being updated during the insertion of a chunk.*

Consider one of these data structures, for instance $A^{H*}B_{<i}$. It can be expressed in database terminology as follows: We start with the binary relation A and select the tuples where the first attribute corresponds to a high-degree vertex in L_1, denoted as A^{H*}. For

the binary relation B, we filter the tuples to include only those belonging to the first $i-1$ chunks, denoted as $B_{<i}$. We then perform a join between the filtered relations A^{H*} and $B_{<i}$ on their shared attribute (vertices in L_2). Finally, we aggregate the result over pairs of vertices in L_1 and L_3, by computing the count of two-hop paths between them.

Data Structures for High vertices. We maintain the following data structures for high degree vertices:

$$A^{H*} \cdot B_{<i}\ ,\quad A^{H*} \cdot B_{<i} \cdot C^{*H} \text{ and } \quad B_{<i} \cdot C^{*H}. \tag{7.1}$$

Claim 7.3.3. *We can maintain $A^{H*} \cdot B_i$ and $B_i \cdot C^{*H}$ on the fly in worst-case update time $O(m^{1/3+\varepsilon_1})$.*

Proof. Consider the data structure $A^{H*} \cdot B_i$ (the analysis for $B_i \cdot C^{*H}$ is symmetric). When we get an edge update (u, v) in B_i we iterate over all $m^{1/3+\varepsilon_1}$ vertices in L_1^H and update the data structure. ∎

Claim 7.3.4. *Multiplying $(A^{H*} \cdot B_i)$ with C^{*H} can be done in time $m^{\omega(1/3+\varepsilon_1, 2/3-\varepsilon_1, 1/3+\varepsilon_1)}$.*

Proof. We know that B_i has at most $m^{2/3-\varepsilon_1}$ edges. Thus, the dimension of $(A^{H*} \cdot B_i)$ is $m^{1/3+\varepsilon_1} \times m^{2/3-\varepsilon_1}$ and the dimension of C^{*H} is $m^{2/3-\varepsilon_1} \times m^{1/3+\varepsilon_1}$. This is because we only have to consider vertices in L_3 that have a non-zero degree, giving us a dimension of $m^{2/3-\varepsilon_1}$ instead of $m^{2/3+2\varepsilon}$. Thus, using rectangular fast matrix multiplication we can multiply $(A^{H*} \cdot B_i)$ with C^{*H} in time $m^{\omega(1/3+\varepsilon_1, 2/3-\varepsilon_1, 1/3+\varepsilon_1)}$. ∎

We need to compute this data structure for every chunk in the time that the next chunk arrives. This gives us a constraint $m^{\omega(1/3+\varepsilon_1, 2/3-\varepsilon_1, 1/3+\varepsilon_1)} \leqslant m^{4/3-2\varepsilon_1}$ which implies:

$$\omega(1/3+\varepsilon_1, 2/3-\varepsilon_1, 1/3+\varepsilon_1) \leqslant 4/3 - 2\varepsilon_1. \tag{7.2}$$

Data Structures for Medium vertices. We maintain the following data structures for medium degree vertices:

$$A^{M*} \cdot B_{<i} \text{ and } \quad B_{<i} \cdot C^{*M}. \tag{7.3}$$

Claim 7.3.5. *We can maintain $A^{M*} \cdot B_i$ and $B_i \cdot C^{*M}$ in time $O(m^{2/3-\varepsilon_1})$.*

Proof. Consider the data structure $A^{M*} \cdot B_i$ (the analysis for $B_i \cdot C^{*M}$ is symmetric). When we get an edge update (u, v) in B_i we iterate over all $m^{2/3-\varepsilon_1}$ vertices in L_1^M and update the data structure. ∎

Data Structures for Low vertices. Consider the low degree vertices in L_1 (analogous for L_4). A^{L*} is the submatrix of A only incident on low degree vertices in L_1. We compute the following data structures:

$$A^{L*} \cdot B_{<i,DD}\ ,\ \ A^{L*} \cdot B_{<i,SS}\ ,\ \ A^{L*} \cdot B_{<i,SD} \text{ and } \ B_{<i,DD} \cdot C^{*L},\ \ B_{<i,SS} \cdot C^{*L},\ \ B_{<i,DS} \cdot C^{*L}. \tag{7.4}$$

Claim 7.3.6. *Computing $A^{L*} \cdot B_{i,DD}$ can be done in time $m^{\omega(2/3+2\varepsilon,1/3-\varepsilon_1+\varepsilon_2,1/3-\varepsilon_1+\varepsilon_2)}$.*

Proof. B_i has at most $m^{1/3-\varepsilon_1+\varepsilon_2}$ dense vertices in L_2 and L_3 and the total number of vertices in L_1 is at most $m^{2/3+2\varepsilon}$. We need to multiply the matrix A^{L*} of dimension $m^{2/3+2\varepsilon} \times m^{1/3-\varepsilon_1+\varepsilon_2}$ with the matrix $B_{i,DD}$ of dimension $m^{1/3-\varepsilon_1+\varepsilon_2} \times m^{1/3-\varepsilon_1+\varepsilon_2}$. Note that the dimension of A^{L*} is $m^{2/3+2\varepsilon} \times m^{2/3+2\varepsilon}$, but we restrict ourselves to the non-zero degree vertices in L_2 when considering the edges of $B_{i,DD}$. We use rectangular fast matrix multiplication for this. Thus, the time taken for fast matrix multiplication is $m^{\omega(2/3+2\varepsilon,1/3-\varepsilon_1+\varepsilon_2,1/3-\varepsilon_1+\varepsilon_2)}$. ∎

This gives us a constraint $m^{\omega(2/3+2\varepsilon,1/3-\varepsilon_1+\varepsilon_2,1/3-\varepsilon_1+\varepsilon_2)} \leqslant m^{4/3-2\varepsilon_1}$ which implies:

$$\omega(2/3+2\varepsilon, 1/3-\varepsilon_1+\varepsilon_2, 1/3-\varepsilon_1+\varepsilon_2) \leqslant 4/3-2\varepsilon_1. \tag{7.5}$$

Claim 7.3.7. *Computing $A^{L*} \cdot B_{i,SS}$ and $A^{L*} \cdot B_{i,SD}$ can be done in time $m^{4/3+\varepsilon_1-\varepsilon_2}$.*

Proof. Go over all $m^{2/3+2\varepsilon}$ low vertices in L_1 and then go over their neighbors in L_2 that are sparse in B_i and then go over their neighbors in L_3 and update the matrix product. The degree of the low vertices in L_1 is at most $m^{1/3+\varepsilon_1}$ and the sparse vertices of $B_{i,L*}$ in L_2 have degree at most $m^{1/3-\varepsilon_2}$ in B_i. Thus, the total time taken is $m^{4/3+\varepsilon_1-\varepsilon_2+2\varepsilon}$. ∎

This gives us a constraint $m^{4/3+\varepsilon_1-\varepsilon_2+2\varepsilon} \leqslant m^{4/3-2\varepsilon_1}$ which implies:

$$3\varepsilon_1 + 2\varepsilon \leqslant \varepsilon_2. \tag{7.6}$$

Proof of Lemma 7.3.2. Claims 7.3.3 to 7.3.7 together prove the statement in Lemma 7.3.2 for B_i. We now want to extend this to hold for $B_{<i}$.

We compute these data structures for each new chunk B_j. Therefore, by induction, when computing the data structure for B_{i-1} we already have the data structure for $B_{<i-1} =$

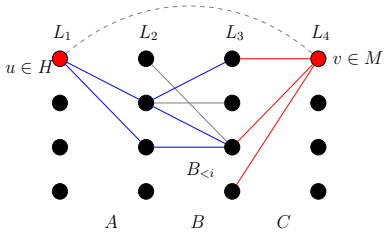


Figure 7.2: This figure shows a query (u, v) where $u \in H$ and $v \in M$. We iterate over the neighbors of v and then use the wedges to u stored in the data structure $A^{H*} \cdot B_{<i}$. The neighbors of v are shown using red edges, the edges in $B_{<i}$ are in gray and the wedges to u are shown in blue.

$\sum_{j=1}^{i-2} B_j$. When we are done computing the data structure for B_{i-1} we add it to the one of $B_{<i-1}$ and get the data structure for $B_{<i} = \sum_{j=1}^{i-1} B_j$. Observe that for this to work we need A and C to remain the same. ∎

7.3.3 Queries:

We have assumed that queries only come in layer D. Let the current query be $e = (u, v)$ where u is in L_1 and v is in L_4. Let B_{i+1} be the current chunk in B that was being inserted when we got query e. Note that the chunk B_{i+1} might be incomplete.

We first describe the procedure of *lazy evaluation* that we use for chunks B_i and B_{i+1}. Given the query u, v we go over all edges of B_i one by one and check if it forms a path from u to v (similar for B_{i+1}). This takes $O(1)$ time per edge in B_i. Thus, we know the number of paths from u to v passing through B_i and B_{i+1}. We now have to only look at paths that go through $B_{<i}$. We will do that in different cases below.

Lemma 7.3.8. *All types of queries can be answered in worst-case time* $O(m^{2/3-\varepsilon_1})$.

Proof. We will divide all types of queries into three cases.

Case 1: HH. In this case, both u, v are high degree vertices.

We know the number of paths from u to v that go through $A, B_{<i}$ and C using the data structure $A^{H*} \cdot B_{<i} \cdot C^{*H}$ (Eq (7.1)). The total time taken is $O(1)$.

Case 2: HM, ML, HL, MM. In this case, $u \in L_1$ is a high or medium degree vertex and $v \in L_4$ is a medium or low degree vertex. Note that this also solves the symmetric cases.

To calculate the number of paths from u to v that go through $B_{<i}$ we can go over the $2m^{2/3-\varepsilon_1}$ neighbors of v in L_3 and use the data structure $A^{H*} \cdot B_{<i}$ ($A^{M*} \cdot B_{<i}$) maintained in Eq (7.1) (Eq (7.3)) to get the number of paths to u. The total time taken is $O(m^{2/3-\varepsilon_1})$. Figure 7.2 gives an illustration of an HM query.

Case 3: LL. In this case, both u, v are low degree vertices.

The low degree vertices can be tricky because we do not have $A^{L*} \cdot B_{<i,DS}$ stored. We iterate over the $2m^{1/3+\varepsilon_1}$ neighbors of v and then compute the number of DD, SS, SD paths using the data structures $A^{L*} \cdot B_{<i,DD}$, $A^{L*} \cdot B_{<i,SS}$, $A^{L*} \cdot B_{<i,SD}$ (Eq (7.4)). For the number of DS paths we calculate the paths from the other side in the following way. We iterate over the $2m^{1/3+\varepsilon_1}$ neighbors of u and use the data structure $B_{i,DS} \cdot C^{*L}$ (Eq (7.4)). The total time taken is $O(m^{1/3+\varepsilon_1})$. ∎

We make a brief remark here that will be addressed now and will not be mentioned for the remainder of the paper. Consider a situation where an edge e is inserted in chunk B_1 and subsequently deleted in chunk B_2. At first glance, this may seem like a problem when creating data structures for chunk B_2, and we may have to go back and change the data structure for chunk B_1 incurring a huge cost. However, this situation can be more effectively understood by treating e as a "negative edge" in chunk B_2. This approach eliminates the need to alter the data structure for B_1 entirely. Once we add the data-structures of B_1 and B_2 together, the entire contribution of edge e will disappear.

For example, consider the data structure $A^{M*}B_i$ and let x be a vertex in L_1^M and y be a vertex in L_3. Let x and y both have edges in B_1 to vertices w and z in L_2. Thus, the data structure $A^{M*}B_1$ will have a value of 2 in position (x, y). Now assume the edge (w, y) is deleted in B_2. Then the data structure $A^{M*}B_2$ will have a value of -1 in position (x, y). Summing up $A^{M*}B_1$ and $A^{M*}B_2$, we get a value of 1 indicating one path between x and y in $A^{M*}B_{\leqslant 2}$ which is the correct answer.

Also, note that the computation required for an insertion or deletion is the same. One concrete way to take care of this problem is to have 8 copies of every data structure because edges inside matrices A, B and C either positive or negative. Another approach is to maintain a single copy, updating it by adding and subtracting the appropriate values, which may sometimes result in negative values. In any case, the time complexity increases by at most an $O(1)$ factor.

7.3.4 Constraints:

The simplest constraints come from the fact that the thresholds for the classes should be in increasing order. So we have $1/3 + \varepsilon_1 \leqslant 2/3 - \varepsilon_1$ and $1/3 - \varepsilon_2 \leqslant 2/3 - \varepsilon_1$ implying the

following constraints:

$$\varepsilon_1 \leqslant 1/6. \tag{7.7}$$

$$\varepsilon_1 - \varepsilon_2 \leqslant 1/3. \tag{7.8}$$

Solving all these constraints (Eq (7.2), Eq (7.5), Eq (7.6), Eq (7.7), Eq (7.8)) gives us $\varepsilon_1 = 0.04201965$ and $\varepsilon_2 = 0.14568075$ when $\varepsilon = 0.0098109$. Thus, we get $\varepsilon_1 \geqslant \varepsilon$. We need this because we use this algorithm as a subroutine in our main algorithm which has update time $O(m^{2/3-\varepsilon})$. Lemma 7.3.2 and Lemma 7.3.8 along with $\varepsilon_1 \geqslant \varepsilon$ prove Lemma 7.3.1. Solving all these constraints for the best possible value of $\omega(a, b, c)$ gives us $\varepsilon_1 = 1/24 = 0.0416666$ and $\varepsilon_2 = 5/24 = 0.2083333$ when $\varepsilon = 1/24$. The best possible value of $\omega(a, b, c)$ is $\max(a + b, b + c)$ which means that the time it takes to multiply the matrices is asymptotically the same as the time it takes to read the input. Section C.2 shows that these values of ε_1 and ε_2, given the value of ε, satisfy all the constraints.

7.4 Setup for the Main Algorithm

In this section, we setup the problem we want to solve. We will eventually prove the following theorem:

Theorem 14. *There is an algorithm for counting the number of layered 4-cycles in any fully dynamic 4-layered graph in worst-case update time $O(m^{2/3-\varepsilon})$ where m is the number of edges in the graph at that point and $\varepsilon > 0$ is a constant. Using the current best value of the square matrix multiplication exponent $\omega = 2.371339$ we get $\varepsilon = 0.009811$ and if we assume the best possible exponent i.e. $\omega = 2$ then we get $\varepsilon = 1/24$.*

The setup will be very similar to the warm-up algorithm in Section 7.3. We let the worst-case update time be $O(m^{2/3-\varepsilon})$ and we will show we can get $\varepsilon > 0$ to be a constant. We will discuss the data structures we need to store and then show how the queries can be answered using them in time $O(m^{2/3-\varepsilon})$. Before we discuss the data structures we need, we group the vertices into different classes.

The vertices in L_1 and L_4 are grouped into classes L, M, H based on their degree in A and C respectively.

- **High** (H): degree between $m^{2/3-\varepsilon}$ and n.

- **Medium** (M): degree between $m^{1/3+\varepsilon}$ and $2m^{2/3-\varepsilon}$.
- **Low** (L): degree between 0 and $2m^{1/3+\varepsilon}$.

We group vertices in L_2 (resp. L_3) into classes S, D based on their combined degree in A, B (resp. B, C).

- **Dense** (D): degree between $m^{2/3-\varepsilon}$ and n.
- **Sparse** (S): degree between 0 and $2m^{2/3-\varepsilon}$.

The classes for vertices in L_2, L_3 have same names here and in Section 7.3.1 but their definitions are very different. The difference is that in the warm-up algorithm we are looking at the degree inside a chunk B_i (there are no chunks here), and here we are looking at the combined degree in A, B (or B, C). Also, note that this algorithm does not have chunks, but it will use the algorithm in Section 7.3 as a subroutine.

Notice that these ranges for different classes (in all layers) overlap with each other, so when a vertex is in an overlapping region it belongs to both classes. The overlapping region will be helpful for vertices that transition from one class to the other.

Finally, we talk about the parameters we will use. The update time of the algorithm will be proved to be $O(m^{2/3-\varepsilon})$ where ε is a parameter. The goal is to show that $\varepsilon > 0$ is a constant. The algorithm will progress in phases of $m^{1-\delta}$ edge updates where $\delta > 0$ is a constant. Lastly, ω is the exponent of square matrix multiplication. These parameters will satisfy the following constraints.

The first constraint we want comes from being able to multiply two square matrices of dimension $m^{2/3+2\varepsilon}$ in a phase. The matrices we are interested in multiplying together are A, B, C each of which have dimension $m^{2/3+2\varepsilon}$. The reason we do this is so that we can get all types of paths between a pair of vertices which belong to the old phases i.e. phases for which this matrix multiplication has been done. The update time for an entire phase is $m^{1-\delta} \cdot m^{2/3-\varepsilon}$ so the constraint we get is $m^{1-\delta} \cdot m^{2/3-\varepsilon} \geqslant m^{\omega \cdot (2/3+2\varepsilon)}$ which implies:

$$1 - \delta \geqslant (2\omega + 1) \cdot \varepsilon + (\omega - 1)2/3. \tag{7.9}$$

This constraint will be used in Section 7.5.1.

The second constraint we want comes from wanting to iterate over pairs of high (or dense) vertices one of which is in a new phase and the other one can be in any of the old

phases. The number of high degree vertices in the new phase is $2m^{1-\delta}/m^{2/3-\varepsilon}$ and the number of high degree vertices in any of the old phases can be at most $m/m^{2/3-\varepsilon}$. The product of the two should be at most the update time $O(m^{2/3-\varepsilon})$ so the constraint we get is $m^{1/3+\varepsilon} \cdot m^{1-\delta-2/3+\varepsilon} \leqslant m^{2/3-\varepsilon}$ which implies:

$$3\varepsilon \leqslant \delta. \tag{7.10}$$

This constraint will be used in Section 7.5.2 and Section 7.5.3.

The simplest constraints come from the fact that the thresholds for the classes should be in increasing order. So we have $1/3 + \varepsilon \leqslant 2/3 - \varepsilon$ implying the following constraint:

$$\varepsilon \leqslant 1/6. \tag{7.11}$$

Solving these equations with the *current best* value of the matrix multiplication exponent $\omega = 2.371339$ gives us $\varepsilon = 0.0098109$ and $\delta = 3\varepsilon = 0.0294327$. Using the *best possible* value of the matrix multiplication exponent $\omega = 2$ gives us $\varepsilon = 1/24$ and $\delta = 1/8$. Section C.2 shows that these values of ε and δ satisfy all the constraints.

7.5 Main Algorithm

In this section, we prove Theorem 14 under a few assumptions.

Assumption 3. *The number of vertices in each layer is $n \leqslant m^{2/3+2\varepsilon}$.*

Assumption 4. *The vertices do not change classes throughout the algorithm.*

For vertices in an overlapping region i.e. they belong to two classes, put them in one of the classes arbitrarily. We will prove the following lemma:

Lemma 7.5.1. *Theorem 14 holds under Assumption 3 and Assumption 4.*

High Level Idea. We would like to do something similar to the previous algorithm (Section 7.3) but here A and C are not fixed so there is a problem with aggregation of data structures over different *chunks* when using the previous approach. For instance, consider the data structure $A^{L*} \cdot B_{<i,DD}$ we maintained in Section 7.3. We maintained it by computing the matrix product of A^{L*} with $B_{j,DD}$ for all $j < i$ and keeping a running sum. The running sum was updated in the following way: $A^{L*} \cdot B_{\leqslant i,DD} = A^{L*} \cdot B_{<i,DD} + A^{L*} \cdot B_{i,DD}$. Doing this is not possible if the matrix A is undergoing changes.

Thus, we create different data structures for vertices in H, M, L that can be updated on the fly i.e. that can be updated during each edge update which we show in Section 7.5.2. However, this is not enough, and we need one more idea to solve the problem. Just like in Section 7.3, we need to use fast matrix multiplication to get a speedup. We divide the edge updates into *phases* of a certain size and during a phase we compute paths for the edges in the old phase whose edges are now fixed, in a way similar to what we did for *chunks*. We explain the idea of phases and its difference from chunks in Section 7.5.1. Lastly, we show how we use these data structures and what we compute for different phases to answer queries in Section 7.5.3.

7.5.1 Phases

We define a **phase** as $m^{1-\delta}$ edge updates. A phase should be long enough so that in the time it takes to process all the edge updates in a phase, we are able to multiply two square matrices of dimension $m^{2/3+2\varepsilon}$. In other words, the number of edges in a phase times the worst-case update time should be more than the time it takes to multiply two square matrices of dimension $m^{2/3+2\varepsilon}$ using fast matrix multiplication. The amount of time it takes to do the multiplication using fast matrix multiplication is $m^{\omega\cdot(2/3+2\varepsilon)}$. Thus, we need $m^{1-\delta} \cdot m^{2/3-\varepsilon} \geqslant m^{\omega\cdot(2/3+2\varepsilon)}$ which we get from Eq (7.9). We divide the incoming edge updates into phases $P_1, P_2, \ldots$ each containing $m^{1-\delta}$ edge updates. This means that the number of vertices that belong to a certain class reduces if we restrict ourselves to a particular phase. For instance, the number of high degree vertices (degree at least $m^{2/3-\varepsilon}$) in a phase is at most $m^{1-\delta-2/3+\varepsilon}$ as opposed to $m^{1-2/3+\varepsilon}$.

A *phase* feels similar to a *chunk* (introduced in Section 7.3.1), but they have slightly different purposes apart from having different sizes. A chunk is small enough for us to do lazy evaluation on and when it is being inserted we create some specific data structures for the previous chunk. During a phase we can compute all types of paths between all pairs of vertices for the previous phase (we are not limited to just specific data structures).

Note that because of the way a phase is set up, during a phase we can compute all matrix products (for relevant submatrices) for the previous phase (for A, B, C, D in that phase). This gives us the number of paths between all pairs of vertices that go through certain classes of vertices (the exact types of paths we need will be clear when we discuss how we answer queries).

We divide all the phases into two parts P_{new} which contains the current phase P_{j+1} and the phase just before it P_j. All phases older than that are represented by P_{old}. The edges in A that are part of P_{old} are represented by A_{old} and those that are part of P_{new} are

represented by A_{new} (similar for B, C). So the new phase has at most $2m^{1-\delta}$ edges but the old phase can have $O(m)$ edges.

A noteworthy observation is that we require $\delta > 0$, which happens only when $\omega < 2.5 - O(\varepsilon)$ (Eq (7.9)) which in particular happens only when $\omega < 2.5$. As we mention in the introduction, this is very surprising because any upper bound on ω better than 3 is not sufficient.

7.5.2 Data Structures

Data Structures			
High (Eq (7.14))	**Medium** (Eq (7.14))	**Low** (Eq (7.13))	**Common** (Eq (7.12))
$A^{HD} \cdot B^{DD}$	$A^{MD} \cdot B^{DD}$	$A^{*D}_{new} \cdot B^{DD}_{old}$	$A^{*S} \cdot B^{S*}$
$B^{DD} \cdot C^{DH}$	$B^{DD} \cdot C^{DM}$	$B^{DD}_{old} \cdot C^{D*}_{new}$	$B^{*S} \cdot C^{S*}$
Additional Data Structures for High (Eq (7.15))			
$A^{HS}_{new} \cdot B^{SS}_{new} \cdot C^{SH}_{new}$, $A^{HS}_{new} \cdot B^{SS}_{new} \cdot C^{SH}_{old}$, $A^{HS}_{old} \cdot B^{SS}_{new} \cdot C^{SH}_{new}$, $A^{HS}_{new} \cdot B^{SS}_{old} \cdot C^{SH}_{new}$, $A^{HS}_{new} \cdot B^{SS}_{old} \cdot C^{SH}_{old}$, $A^{HS}_{old} \cdot B^{SS}_{old} \cdot C^{SH}_{new}$			

Table 7.2: Data Structures used in the Main Algorithm.

We discuss the data structures we store to answer the queries in $O(m^{2/3-\varepsilon})$ time. We will show that the worst-case time it takes to update the data structures is $O(m^{2/3-\varepsilon})$. The list of data structures can be found in Table 7.2.

Lemma 7.5.2. *The worst-case time it takes to update all the data structures we store is* $O(m^{2/3-\varepsilon})$.

Consider one of these data structures, for instance $A^{HS}_{new} \cdot B^{SS}_{new} \cdot C^{SH}_{old}$. It can be expressed in database terminology as follows: We start with the binary relation A and filter the tuples to include only those belonging to the new phase. We then select the tuples where the first attribute corresponds to a high-degree vertex in L_1 and the second attribute corresponds to a sparse vertex in L_2, denoted as A^{HS}_{new}. For the binary relation B, we similarly filter the tuples to include only those from the new phase and then select the tuples where both attributes correspond to sparse vertices in L_2 and L_3, denoted as B^{SS}_{new}. For the binary relation C, we filter the tuples to include only those from the old phase and then select the tuples where the first attribute corresponds to a sparse vertex in L_3 and the second to a high-degree vertex in L_4, denoted as C^{SH}_{old}. We then perform a join between the filtered

relations A^{HS}_{new}, B^{SS}_{new}, and C^{SH}_{old} on their shared attributes (vertices in L_2 and L_3), forming three-hop paths. Finally, we aggregate the result over pairs of vertices in L_1 and L_4, by computing the count of three-hop paths between them.

Common Data Structures. We first discuss the data structures we store for all the vertices. We maintain the data structures:

$$A^{*S} \cdot B^{S*} \text{ and } \; B^{*S} \cdot C^{S*}. \tag{7.12}$$

Claim 7.5.3. *The worst-case update time for the data structures $A^{*S} \cdot B^{S*}$ and $B^{*S} \cdot C^{S*}$ is $O(m^{2/3-\varepsilon})$.*

Proof. Consider the data structure $A^{*S} \cdot B^{S*}$. The analysis for the other is identical. If there is an update (u, v) in A^{*S} then we go over the $2m^{2/3-\varepsilon}$ neighbors of v in L_3 and update the number of paths in the data structure. If the update (u, v) is in B^{S*} then we iterate over the $2m^{2/3-\varepsilon}$ neighbors of u in L_1 and update the number of paths. ∎

We also store the following data structures for all vertices, but we use them only for low degree vertices:

$$A^{*D}_{new} \cdot B^{DD}_{old} \text{ and } \; B^{DD}_{old} \cdot C^{D*}_{new}. \tag{7.13}$$

Claim 7.5.4. *The worst-case update time for data structures $A^{*D}_{new} \cdot B^{DD}_{old}$, $B^{DD}_{old} \cdot C^{D*}_{new}$ is $O(m^{1/3+\varepsilon})$.*

Proof. Observe that we only deal with B_{old} and not B_{new} so there are no updates in B. If there is an update (u, v) in A (or C) then we iterate over the dense neighbors of v in L_3 in $O(m^{1/3+\varepsilon})$ time and update the data structure. ∎

Data Structures for vertices in H,M. We now mention the data structures we store for the high and medium degree vertices in L_1 and L_4. We maintain the following data structures:

$$A^{HD} \cdot B^{DD}, \;\; B^{DD} \cdot C^{DH} \text{ and } \; A^{MD} \cdot B^{DD}, \;\; B^{DD} \cdot C^{DM}. \tag{7.14}$$

Notice that we can always update these data structures for any edge update in $O(m^{2/3-\varepsilon})$ time. The reason for this is that the maximum size of these restricted layers is at most $m^{2/3-\varepsilon}$.

Claim 7.5.5. *The worst-case update time for the data structures in Eq (7.14) is $O(m^{2/3-\varepsilon})$.*

Proof. Consider the data structure $A^{MD} \cdot B^{DD}$. If there is an update (u, v) in A^{MD} then we go over the $m^{1/3+\varepsilon}$ neighbors of v in L_3^D and update the number of paths in the data structure. If the update is in B^{DD} then we iterate over the $m^{2/3-\varepsilon}$ vertices in L_1^M and update the number of paths. The analysis is similar for $B^{DD} \cdot C^{DM}$. The same arguments work for high degree vertices. This concludes the proof. ∎

We maintain six additional data structures for high degree vertices:

$$\begin{aligned} &A_{new}^{HS} \cdot B_{new}^{SS} \cdot C_{new}^{SH}, \quad A_{new}^{HS} \cdot B_{new}^{SS} \cdot C_{old}^{SH}, \quad A_{old}^{HS} \cdot B_{new}^{SS} \cdot C_{new}^{SH}, \\ &A_{new}^{HS} \cdot B_{old}^{SS} \cdot C_{new}^{SH}, \quad A_{new}^{HS} \cdot B_{old}^{SS} \cdot C_{old}^{SH}, \quad A_{old}^{HS} \cdot B_{old}^{SS} \cdot C_{new}^{SH}. \end{aligned} \tag{7.15}$$

This list includes all possible eight combinations of old and new phases *other than* $A_{old}^{HS} \cdot B_{new}^{SS} \cdot C_{old}^{SH}$ and $A_{old}^{HS} \cdot B_{old}^{SS} \cdot C_{old}^{SH}$.

Claim 7.5.6. *The worst-case update time for the additional data structures for high degree vertices in Eq (7.15) is $O(m^{2/3-\varepsilon})$.*

Proof. We can store the data structures of the form $A^{HS} \cdot B^{S*}$ and $B^{*S} \cdot C^{SH}$ (Eq (7.12)) for all combinations of old and new phases using the same proof as Claim 7.5.3.

Say we get an edge update in A (resp. C) then we can go over all vertices $m^{1/3+\varepsilon}$ in L_4^H (resp. L_1^H) and update the number of paths. This is possible since we have the number of $B^{*S} \cdot C^{SH}$ (resp. $A^{HS} \cdot B^{S*}$) paths stored.

If we get an edge update in B then we iterate over all pairs of vertices in L_1^H and L_4^H and update the data structure. This seems like it will take too much time on the surface, but we know that the data structure has either A or C in the new phase and hence the number of high degree vertices in a new phase is at most $m^{1-\delta-2/3+\varepsilon}$ as opposed to $m^{1-2/3+\varepsilon}$. Thus, iterating over all pairs of high vertices takes time $m^{1/3+\varepsilon} \cdot m^{1-\delta-2/3+\varepsilon} \cdot O(1) \leqslant O(m^{2/3-\varepsilon})$ (Eq (7.10)). ∎

Proof of Lemma 7.5.2. Claims 7.5.3 to 7.5.6 together prove Lemma 7.5.2. ∎

7.5.3 Queries

We now discuss how we can answer queries in worst-case time $O(m^{2/3-\varepsilon})$. We will prove the following:

Lemma 7.5.7. *The worst-case query time is $O(m^{2/3-\varepsilon})$.*

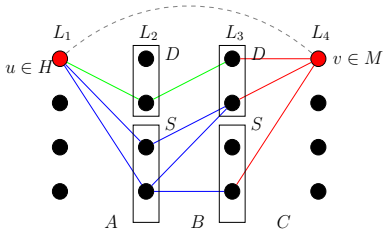


Figure 7.3: This figure shows a query (u, v) where $u \in H$ and $v \in M$. We iterate over the neighbors of v (shown in red) and then use the wedges through sparse vertices to u stored in the data structure $A^{*S}B^{S*}$ (shown in blue). To get the DD paths we use the wedges through dense vertices to u stored in the data structure $A^{HD}B^{DD}$ (shown in green). To get the DS paths we need to iterate over the neighbors of u in L_2^D and use the wedges through sparse vertices to v stored in the data structure $B^{*S}C^{S*}$ (not shown here).

Let the current query be $e = (u, v)$ where u is in L_1 and v is in L_4. We first discuss the queries when the endpoints belong to H or M.

Claim 7.5.8. *The worst-case query time when both the query endpoints belong to H or M is $O(m^{2/3-\varepsilon})$.*

Proof. There are two choices for vertices in L_2 and L_3, sparse or dense, so there are four kinds of paths between any pair of vertices in L_1 and L_4. Consider paths where there is at least one dense vertex. We can iterate over all the $m^{1/3+\varepsilon}$ dense vertices in that layer and check for an edge to one side and then the number of paths to the other side. This can be done using the data structures of the form $A^{HD} \cdot B^{DD}$ (Eq (7.14)) for paths through dense vertices and $A^{*S} \cdot B^{S*}$ (Eq (7.12)) for paths through sparse vertices.

So now we only worry about paths that go through L_2^S and L_3^S. If one of the vertices in the query is in M, let it be in L_1 without loss of generality, then we iterate over its $2m^{2/3-\varepsilon}$ neighbors in L_2^S and use $B^{*S} \cdot C^{S*}$ (Eq (7.12)) to get the number of paths to the other endpoint of the query that go through L_3^S.

The remaining case is where the query has both high degree endpoints, and we need to calculate the number of SS paths. We know the number of $A_{old} \cdot B_{old} \cdot C_{old}$ paths from the matrix product of the previous phase. We also know the number of $A_{old} \cdot B_{new} \cdot C_{old}$ paths because we can run the algorithm where A, C are fixed (Section 7.3). For the remaining combinations of phases, we have data structures of the form $A^{HS} \cdot B^{SS} \cdot C^{SH}$ (Eq (7.15)) where the number of paths are explicitly stored. Figure 7.3 gives an illustration of an HM query. ∎

We now discuss the queries when at least one endpoint belong to L.

Claim 7.5.9. *The worst-case query time when at least one of the query endpoints belongs to L is $O(m^{2/3-\varepsilon})$.*

Proof. We first consider HL and ML queries. In this case, $u \in L_1$ is a high or medium degree vertex and $v \in L_4$ is a low degree vertex. We can iterate over all $2m^{1/3+\varepsilon}$ neighbors w_i of v. We know the number of DD, SS, SD paths from u to w_i's using the data structures $A^{*S} \cdot B^{S*}$ (Eq (7.12)) and $A^{HD} \cdot B^{DD}, A^{MD} \cdot B^{DD}$ (Eq (7.14)). For the DS paths we iterate over the $2m^{1/3+\varepsilon}$ neighbors of u in L_2^D and use $B^{*S} \cdot C^{S*}$ (Eq (7.12)) to get the number of paths to v via L_3^S.

We now consider LL queries. In this case, $u \in L_1^L$ and $v \in L_4^L$ are both low degree vertices. It is easy to find the number of SS, SD, DS paths by going over the neighbors of a query vertex and then using the data structures $A^{*S} \cdot B^{S*}, B^{*S} \cdot C^{S*}$ (Eq (7.12)). So we now only worry about DD paths. Consider the different cases depending on the phases A, B and C are in:

Case 1: $A_{old} \cdot B_{old} \cdot C_{old},\ A_{old} \cdot B_{old} \cdot C_{new},\ A_{new} \cdot B_{old} \cdot C_{old}$
Consider $A_{old} \cdot B_{old} \cdot C_{new}$ paths (the others are similar). We can iterate over the $2m^{1/3+\varepsilon}$ neighbors of v and use the number of $A_{old} \cdot B_{old}$ paths stored in the matrix product for the old phase.

Case 2: $A_{new} \cdot B_{old} \cdot C_{new}$
We have the following data structures stored: $A_{new}^{LD} \cdot B_{old}^{DD}, B_{old}^{DD} \cdot C_{new}^{DL}$ (Eq (7.13)). Iterate over the $2m^{1/3+\varepsilon}$ neighbors of u and use the number of DD paths to v.

Case 3: $A_{new} \cdot B_{new} \cdot C_{new},\ A_{new} \cdot B_{new} \cdot C_{old},\ A_{old} \cdot B_{new} \cdot C_{new}$
Either L_2 or L_3 has just $O(m^{1-\delta})$ edges incident on it. Thus, we go over all pairs of dense vertices in L_2, L_3 in time $O(m^{1/3+\varepsilon}) \cdot O(m^{1/3-\delta+\varepsilon}) \leqslant O(m^{2/3-\varepsilon})$ (Eq (7.10)) and check if they form a path.

Case 4: $A_{old} \cdot B_{new} \cdot C_{old}$
We can run the algorithm where A, C are fixed (Section 7.3).

Claim 7.5.8 and Claim 7.5.9 together prove Lemma 7.5.7. Also, Lemma 7.5.2 and Lemma 7.5.7 together prove Lemma 7.5.1. Therefore, we have proved Theorem 14 under Assumption 3 and Assumption 4. The next step is getting rid of these assumptions which we will do in the following sections. The last thing we do in this section is give the pseudo code for the algorithms.

7.5.4 Pseudocode

In this subsection, we give some high level pseudocode for the main algorithm. Algorithm 12 is the main algorithm (Equivalent Queries in Section 7.2) which outputs the total number of 4-cycles in the current graph after every update. It uses Algorithm 13 and Algorithm 14 as subroutines which are algorithms that update the data structures (Section 7.5.2) and answer the query (Section 7.5.3) respectively. Note that these algorithms are only high level and the full details can be found in the referenced sections.

Algorithm 12. **Counting layered 4-Cycles**

Input: A dynamic 4-layered graph G and an edge insertion or deletion (u, v).
Output: Maintain the count of total layered 4-cycles in G.

Step 1: Process Edge Update (u, v)

- **Call Update**(u, v) on the three relevant copies.

Step 2: Compute Number of 4-Cycles

- **Call Query**(u, v) on the fourth copy:

$$\text{New} \leftarrow \textbf{Query}(u, v)$$

- Use the previous query answer to update the total count:

$$\text{Total} \leftarrow \text{Total} + \text{New} \qquad \text{(Insertion)}$$
$$\text{Total} \leftarrow \text{Total} - \text{New} \qquad \text{(Deletion)}$$

Output Total.

Algorithm 13. **Update**(u, v)

Input: A dynamic graph G, an edge update $(u, v) \in B$, where $u \in L_2$ and $v \in L_3$.
Output: Updates only the affected data structures containing B, reflecting the addition of (u, v).

Step 1: Update Affected Data Structures:
If $u, v \in D$ **:**

- Update data structures in Eq (7.14) by iterating over all neighbors of u in L_1^H or L_1^M (similar for neighbors of v in L_4^H, L_4^M).

Else:

- Update data structures in Eq (7.12) by iterating over all neighbors of u (similarly iterate over all

neighbors of v).

- **If** $u, v \in S$ **:** Update data structures in Eq (7.15) by iterating over all pairs of vertices in L_1^H and L_4^H.

Step 2: Matrix Multiplication for the Old Phase:

- If (u, v) is the first edge of a phase:
  We start the computation of matrix products for all relevant submatrices in the old phase.
- Else:
  Continue the matrix multiplication computation for $O(m^{2/3-\varepsilon})$ steps.

Algorithm 14. Query(u, v)

Input: A dynamic graph G, a query (u, v) where $u \in L_1$ and $v \in L_4$.
Output: Number of 3-paths between u and v.

If $u, v \in H \cup M$**:**

- Compute α the number of DD, SD, DS paths by iterating over L_2^D, L_3^D and using data structures of the form $A^{HD} \cdot B^{DD}$ for dense paths and $A^{*S} \cdot B^{S*}$ for sparse paths.
- **If** u and $v \in H$, let β be the sum of these paths:
  - Retrieve precomputed paths in $A_{old} \cdot B_{old}^{SS} \cdot C_{old}$ and data structures in Eq (7.15).
  - Query u, v for the copy of the warm up algorithm run on $A_{old} \cdot B_{new}^{SS} \cdot C_{old}$.
- **Else** (wlog $u \in M$) and let β be the number of following paths. Iterate over neighbor of u in L_2^S and use $B^{*S} \cdot C^{S*}$.

Else If $u, v \in L$**:**

- Compute α the number of SS, SD, DS paths using $A^{*S} \cdot B^{S*}, B^{*S} \cdot C^{S*}$.
- Compute β the number of DD paths based on phases:
  - **Case 1:** $A_{old} \cdot B_{old} \cdot C_{old}, A_{old} \cdot B_{old} \cdot C_{new}, A_{new} \cdot B_{old} \cdot C_{old}$
    Iterate over neighbors in D and use stored paths.
  - **Case 2:** $A_{new} \cdot B_{old} \cdot C_{new}$
    Iterate over neighbors in D and use data structures $A_{new}^{LD} \cdot B_{old}^{DD}$ and $B_{old}^{DD} \cdot C_{new}^{DL}$.
  - **Case 3:** $A_{new} \cdot B_{new} \cdot C_{new}, A_{new} \cdot B_{new} \cdot C_{old}, A_{old} \cdot B_{new} \cdot C_{new}$
    Iterate over all pairs of dense vertices in L_2, L_3 and check paths with u, v.

– **Case 4:** $A_{old} \cdot B_{new} \cdot C_{old}$
Query u, v for the copy of the warm up algorithm run for $A_{old} \cdot B_{new} \cdot C_{old}$.

Else: (wlog $u \in L$)

- Iterate over the neighbors of u.
- Compute α the number of SS, SD, DS paths using $A^{*S} \cdot B^{S*}$ and $B^{*S} \cdot C^{S*}$.
- Compute β the number of DD paths using $A^{HD} \cdot B^{DD}, A^{MD} \cdot B^{DD}$.

Return $\alpha + \beta$.

Note that Algorithm 13 is written only for when the update is in matrix B. We have similar algorithms for when the update is in A, C which update different data structures (using similar ideas). In Step 2 of Algorithm 13 we want to compute the number of paths between all pairs of vertices (between all layers) for all choices of classes of vertices (S, D) in L_2 and L_3. This is done by going over all possible $O(1)$ combinations of submatrices and multiplying the compatible ones using fast matrix multiplication. The matrix multiplications we compute are $A^{*x} \cdot B^{xy} \cdot C^{y*}$, $A^{*x} \cdot B^{x*}$ and $B^{*y} \cdot C^{y*}$ for all choices of $x, y \in \{S, D\}$. In Algorithm 14, α and β are just some counters where α counts 3 types of paths and β counts one type of path (there is no connection between α and β across different cases).

7.6 Getting Rid of Assumption 3: Tiny Vertices

In this section, we introduce **tiny** vertices (T) and show how they help get rid of Assumption 3. The vertices in L_1 (resp. L_4) are in T when their degree in A (resp. C) is at most $2m^{1/3-2\varepsilon}$. The vertices in L_2 (resp. L_3) are in T when their combined degree in A, B (resp. B, C) is at most $2m^{1/3-2\varepsilon}$.

We describe how we calculate the number of paths that go through tiny vertices and also how we answer queries when at least one endpoint is a tiny vertex. Once we do this we can completely ignore tiny vertices. For queries with a tiny vertex we will know how to compute the answer and for other queries we will be able to figure out all the paths that go through tiny vertices, so the only thing left to do will be to calculate the number of paths that do not go through any tiny vertices. This is precisely what we focused on in the previous section Section 7.5 (modulo Assumption 4). We will prove the following lemma:

Lemma 7.6.1. *Theorem 14 holds under Assumption 4.*

High Level Idea. The goal of this section is to say that tiny vertices are easy to handle. We will show that we can easily compute paths through tiny vertices and answer queries

containing tiny vertices. It is easy to maintain 2-paths through tiny vertices, but we also show in Section 7.6.1 that we can store some types of 3-paths through them in worst-case update time $O(m^{2/3-\varepsilon})$. These are helpful for finding paths through tiny vertices when one of the query endpoints is a high degree vertex. If both the endpoints are not high degree then we can easily iterate over the neighbors of each of the query vertices and then use the data structure for wedges through tiny vertices. The complete argument for this part can be found in Section 7.6.3.

Answering queries when one of the endpoints is a tiny vertex is the easier part, and we show the details in Section 7.6.2. If the other vertex is tiny or low degree then we just iterate over the neighbors of both the endpoints. Otherwise, we go over the neighbors of the tiny vertex and use the stored wedges through tiny and sparse vertices. The only remaining case is paths going through dense vertices and the crucial observation is that we can iterate over all the neighbors of a tiny vertex and all the dense vertices of a layer. This turns out to be enough to solve the problem.

7.6.1 Data Structure for Tiny Vertices

Data Structures		
High	**Medium**	**Common**
$A^{HT} \cdot B^{TT} \cdot C^{TH}$ (Eq (7.17))	$A^{MT} \cdot B^{TT} \cdot C^{TH}$ (Eq (7.17))	$A^{*T} \cdot B^{T*}$ (Eq (7.16))
$A^{HT} \cdot B^{TS} \cdot C^{SH}$ (Eq (7.18))	$A^{HT} \cdot B^{TT} \cdot C^{TM}$ (Eq (7.17))	$B^{*T} \cdot C^{T*}$ (Eq (7.16))
$A^{HS} \cdot B^{ST} \cdot C^{TH}$(Eq (7.18))		

Table 7.3: Data Structures for Tiny Vertices.

We show the data structures we need to store to calculate the number of paths through tiny vertices and to answer queries with tiny vertices in them. The list of data structures can be found in Table 7.3.

Lemma 7.6.2. *The worst-case time it takes to update all the data structures we store for tiny vertices is* $O(m^{2/3-\varepsilon})$.

We first maintain the following data structures for tiny vertices:

$$A^{*T} \cdot B^{T*} \text{ and } B^{*T} \cdot C^{T*}. \tag{7.16}$$

Claim 7.6.3. *The worst-case update time for the data structures* $A^{*T} \cdot B^{T*}$ *and* $B^{*T} \cdot C^{T*}$ *is* $O(m^{1/3-2\varepsilon})$.

Proof. The proof is almost identical to the proof of Claim 7.5.3. Consider the data structure $A^{*T} \cdot B^{T*}$. The analysis for the other case is identical. If there is an update (u, v) in A^{*T} then we go over the $2m^{1/3-2\varepsilon}$ neighbors of v in L_3 and update the number of paths in the data structure. If the update (u, v) is in B^{T*} then we iterate over the $2m^{1/3-2\varepsilon}$ neighbors of u in L_1 and update the number of paths. ∎

We maintain some data structures just for high and medium degree vertices:

$$A^{HT} \cdot B^{TT} \cdot C^{TH}, A^{MT} \cdot B^{TT} \cdot C^{TH}, A^{HT} \cdot B^{TT} \cdot C^{TM}. \tag{7.17}$$

Claim 7.6.4. *The worst-case update time for the data structures $A^{HT} \cdot B^{TT} \cdot C^{TH}$, $A^{MT} \cdot B^{TT} \cdot C^{TH}$, $A^{HT} \cdot B^{TT} \cdot C^{TM}$ is $O(m^{2/3-\varepsilon})$.*

Proof. Consider the data structure $A^{HT} \cdot B^{TT} \cdot C^{TM}$. The analysis for the others is identical. First we note that we already have the data structures $A^{*T} \cdot B^{T*}, B^{*T} \cdot C^{T*}$ stored. This gives us access to $A^{HT} \cdot B^{TT}$ and $B^{TT} \cdot C^{TM}$.

If there is an update (u, v) in A^{HT} then we go over the $m^{2/3-\varepsilon}$ vertices in L_4^M and update the number of paths using $B^{TT} \cdot C^{TM}$. Similarly, if there is an update (u, v) in C^{TM} then we go over the $m^{1/3+\varepsilon}$ vertices in L_1^H and update the number of paths using $A^{HT} \cdot B^{TT}$.

Finally, if there is an update (u, v) in B^{TT} then we go over the $2m^{1/3-2\varepsilon}$ neighbors of u in L_1^H and v in L_4^M and update the number of paths in the data structure. The total number of pairs is $4m^{2/3-4\varepsilon}$, so we are done. ∎

We maintain additional data structures for high degree vertices:

$$A^{HT} \cdot B^{TS} \cdot C^{SH}, A^{HS} \cdot B^{ST} \cdot C^{TH}. \tag{7.18}$$

Claim 7.6.5. *The worst-case update time for these data structures for high degree vertices (Eq (7.18)) is $O(m^{2/3-\varepsilon})$.*

Proof. Consider the data structure $A^{HT} \cdot B^{TS} \cdot C^{SH}$. The analysis for $A^{HS} \cdot B^{ST} \cdot C^{TH}$ is identical. First we note that we already have data structures $A^{*T} \cdot B^{T*}$ and $B^{*S} \cdot C^{S*}$ stored. This gives us access to $A^{HT} \cdot B^{TS}$ and $B^{TS} \cdot C^{SH}$.

If there is an update (u, v) in A^{HT} then we go over the $m^{1/3+\varepsilon}$ vertices in L_4^H and update the number of paths using $B^{TS} \cdot C^{SH}$. Similarly, if there is an update (u, v) in C^{SH} then we go over the $m^{1/3+\varepsilon}$ vertices in L_1^H and update the number of paths using $A^{HT} \cdot B^{TS}$.

Finally, if there is an update (u, v) in B^{TS} then we go over the $2m^{1/3-2\varepsilon}$ neighbors of u in L_1^H and also the $m^{1/3+\varepsilon}$ vertices in L_4^H and update the number of paths in the data structure. The total number of pairs is $2m^{2/3-\varepsilon}$, so we are done. ∎

Claims 7.6.3 to 7.6.5 together prove Lemma 7.6.2. We now show how we answer queries that contain tiny vertices.

7.6.2 Queries for Tiny Vertices

Here we show how to answer queries when at least one of the vertices in the query is a tiny vertex.

Lemma 7.6.6. *The worst-case query time when at least one of the vertices in the query is a tiny vertex is $O(m^{2/3-\varepsilon})$.*

Proof. We divide the types of queries into two cases.

Case 1: TT, TL. If one vertex is in T and the other is in T or L then we can iterate over both of their neighbors in time $2m^{1/3-2\varepsilon} \cdot 2m^{1/3+\varepsilon} \cdot O(1) = O(m^{2/3-\varepsilon})$ and then calculate the number of paths.

Case 2: TM, TH. If one vertex is in T and the other is in H, M then we can first calculate the number of paths that go through L_3^D. This is done by iterating over the $2m^{1/3-2\varepsilon}$ neighbors of u and all $m^{1/3+\varepsilon}$ vertices in L_3^D and checking if they form a path. The time taken is $O(m^{2/3-\varepsilon})$.

We can iterate over the $2m^{1/3-2\varepsilon}$ neighbors of u and then use the data structure $B^{*S} \cdot C^{S*}$ (Eq (7.12)) to get the number of paths going through L_3^S and use $B^{*T} \cdot C^{T*}$ (Eq (7.16)) to get the number of paths going through L_3^T. Thus, we get all types of paths ∎

7.6.3 Paths Through Tiny Vertices

Now we need to show that we can easily calculate the number of paths through tiny vertices in L_2 and L_3 no matter what kind of query we get. Once we do this we can ignore tiny vertices for the rest of the analysis.

Lemma 7.6.7. *The worst-case time it takes to calculate the number of paths through tiny vertices in L_2 or L_3 during an arbitrary query is $O(m^{2/3-\varepsilon})$.*

Proof. We first describe queries where there are no high degree vertices. Let the query be u, v. We have the data structures $A^{*T} \cdot B^{T*}$ and $B^{*T} \cdot C^{T*}$ (Eq (7.16)) stored. We iterate over neighbors of u in $O(m^{2/3-\varepsilon})$ time and use $B^{*T} \cdot C^{T*}$ to get the paths that go to v via L_3^T. We can do the same side from the side of v and get the paths through L_2^T. Note that we will include the TT paths just once.

We can now see why the above procedure does not work for high degree vertices. High degree vertices have too many neighbors to iterate over. So we handle these cases separately.

Case 1: HH. We can easily get the number of TT paths using $A^{HT} \cdot B^{TT} \cdot C^{TH}$ (Eq (7.17)) and TS paths using $A^{HT} \cdot B^{TS} \cdot C^{SH}$ (Eq (7.18)). Finally, for the number of TD paths, we iterate over the $m^{1/3+\varepsilon}$ neighbors of v in L_3^D and then use the data structure $A^{*T} \cdot B^{T*}$ (Eq (7.16)) to get the number of paths to u via L_2^T. The other paths can be found symmetrically.

Case 2: HM. We can easily get the number of TT paths using $A^{HT} \cdot B^{TT} \cdot C^{TM}$ (Eq (7.17)). For the number of TD paths (and DT paths) we iterate over the $m^{1/3+\varepsilon}$ neighbors of v in L_3^D and then use the data structure $A^{*T} \cdot B^{T*}$ (Eq (7.16)) to get the number of paths to u via L_2^T. To get the number of ST and TS paths we iterate over the $2m^{2/3-\varepsilon}$ neighbors of v and then use $A^{*S} \cdot B^{S*}$ (Eq (7.12)) to get the number of ST paths and use $A^{*T} \cdot B^{T*}$ (Eq (7.16)) to get the number of TS paths.

Case 3: HL. To find the number of $*T$ paths we can iterate over the neighbors of neighbors of v and check if they have an edge to u. This takes $2m^{1/3+\varepsilon} \cdot 2m^{1/3-2\varepsilon} \cdot O(1) = O(m^{2/3-\varepsilon})$ time. For the number of $T*$ paths we can iterate over the neighbors of v and use $A^{*T} \cdot B^{T*}$ (Eq (7.16)). ∎

This means that we can figure out the number of paths through tiny vertices, and we can also answer queries when they involve tiny vertices, so we can completely ignore tiny vertices in the rest of the analysis. Thus, we can assume that each layer has at most $m^{2/3+2\varepsilon}$ vertices because all the remaining vertices have degree at least $m^{1/3-2\varepsilon}$.

Proof of Lemma 7.6.1. We are given a layered graph with n vertices in each layer, and we are still under Assumption 4. We know from Lemma 7.6.2 that we can maintain the data structures for tiny vertices in $O(m^{2/3-\varepsilon})$ time. We know from Lemma 7.6.6 that answering queries which include tiny vertices can be answered in time $O(m^{2/3-\varepsilon})$. Finally, Lemma 7.6.7 tells us that we can calculate the number of paths through tiny vertices in time $O(m^{2/3-\varepsilon})$.

Thus, the only thing left to do is to answer queries with no tiny vertices and for them only calculate paths that go through no tiny vertices. This means that we can assume that each layer has at most $m^{2/3+2\varepsilon}$ vertices because all the remaining vertices have degree at least $m^{1/3-2\varepsilon}$.

Therefore, we can assume Assumption 3 without loss of generality. Also, we have already proved Lemma 7.5.1. This should be enough to prove Lemma 7.6.1.

We need a final piece here to clarify some details. The common data structures we have in Eq (7.12) and Eq (7.13) currently hold only when $*$ is either H, M or L. We will make a case that they hold even when $*$ is T. This is done by observing that the proof of their update times never used the size of the layers.

Consider the data structure $A^{*S} \cdot B^{S*}$ in Eq (7.12) (the proof for the other one is symmetric). When an edge update in A or B is incident on a vertex $v \in L_2^S$ we go over the $m^{2/3-\varepsilon}$ neighbors of v and update the data structure. Thus, it does not matter what the size of the layers is because the number of neighbors of v is small.

Consider the data structure $A_{new}^{*D} \cdot B_{old}^{DD}$ in Eq (7.13) (the proof for the other one is symmetric). When an edge update in A_{new} (there are no updates in B_{old}) is incident on a vertex $v \in L_2^D$ we go over the $m^{1/3+\varepsilon}$ vertices in L_3^D and update the data structure. Thus, it does not matter what the size of the layers is because the number of vertices in L_3^D is small. Therefore, the common data structures work even when $*$ is T. ∎

7.7 Getting Rid of Assumption 4: Vertices Moving Between Classes

In this section, we get rid of Assumption 4, namely, handle the case when the vertices may move between classes due to the change in their degrees.

7.7.1 Setup

Note that until now we have assumed that the classes of vertices are fixed. But if the degree of vertices change then their classes will change too. So in this section, we describe what happens when the degrees of the vertices changes by a lot, and they have to change classes.

When the degree of a vertex is changing it goes from being in one class to the overlapping region of two classes and then finally into one class again. When the vertex moves into to the overlapping region of two classes we treat it like it still belongs to the old class.

During this period we will create data structures of both classes for this vertex. After undergoing edge insertions and deletions if the vertex ends up in its old class then we do not have to do anything. However, if it ends up in a new class then we have to update the data structures for both the old and new classes. Note that we have to address *two types* of vertex transitions, one where the vertex is the endpoint of a path called **Type-1** transitions and the other where the vertex is an intermediate vertex in a path called **Type-2** transitions.

Type-1 Transition. In this type of transition, the transitioning vertex is the endpoint of a path for some data structure. For instance consider the data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ (Eq (7.17)) for medium vertices that we do not have for low vertices. Consider a vertex $u \in L_1$ with degree $m^{1/3+\varepsilon}$ which after an edge insertion, has just entered the transition range between L and M. Computing this data structure for u means that for all vertices $v_i \in L_4^H$ we need to compute the number of paths between u and v_i that go through L_2^T and L_3^T. This is the first type of transition and what we have to do for it is add a new row to the data structures for medium vertices like $A^{MT} \cdot B^{TT} \cdot C^{TH}$ corresponding to the answers we compute for u. We also need to remove all the rows corresponding to u in the data structures for low vertices.

Type-2 Transition. In this type of transition, the transitioning vertex is an intermediate vertex in a path for some data structure. Consider the same data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ (Eq (7.17)) and consider a vertex $y \in L_2$ with degree $m^{1/3-2\varepsilon}$ which after an edge insertion, has just entered the transition range between T and S. Computing this data structure for y means that for all pairs of vertices $u_i \in L_1^M$ and $v_j \in L_4^H$ we need to compute the number of paths between u_i and v_j that go through y in L_2 and through some vertex in L_3^T. This is the second type of transition and what we have to do for it is update the values in data structures for medium vertices like $A^{MT} \cdot B^{TT} \cdot C^{TH}$ corresponding to the answers we compute for vertex y. So this can be thought of computing a difference matrix of dimension $\left|L_1^M\right| \times \left|L_4^H\right|$ which when added to the matrix $A^{MT} \cdot B^{TT} \cdot C^{TH}$ includes the contribution of vertex $y \in L_2$. We also have to update the values in data structures for low vertices. The major difference between the two types of transitions is that for the first type of transition (Type-1), we need to add or delete a new row of values corresponding to the transitioning vertex u, and we do not have to change any existing values. In the second type of transition (Type-2), we do not have to add or delete any rows but update the existing values in the data structures for all pairs that are affected by the transitioning vertex y.

Note that the computation to create data structures for a vertex v will only be done when we get an edge incident to this vertex v so by the time its degree doubles we have the

new data structure ready for v. This ensures that during an edge update we are doing this computation for at most two vertices and thus does not affect the worst-case update time (if we are not careful about this then the guarantee we get will be in terms of amortized update time). Finally, when the vertex v enters the transition range, we partition the edges into the ones that came before and after that moment and deal with them separately. Let edge e be the update after which vertex v entered the transition range. We represent by E^v_{bef}, the set of edge updates that arrive before edge update e (including e) and by E^v_{aft} set of edge updates that arrive after edge update e. We will drop the superscript of v when the vertex is clear from context. We also denote by A_{bef} and A_{aft} the set of edges in A before and after the edge update e, so we have $A = A_{bef} + A_{aft}$. Note that A_{aft} can have a negative value for some edge because that edge was inserted in A_{bef} and deleted in A_{aft} but this is not going to be a problem. We define similar notation for B and C. The main task is to compute the data structure for the edges that come before the transition moment (E_{bef}) in $O(x \cdot m^{2/3-\varepsilon})$ time, where x is the degree of the vertex when it enters the overlapping region.

Consider any data structure we need to compute for the new class where edges of E_{aft} are being used (in A, B or C) for instance consider $A^{MT}_{aft} \cdot B^{TT}_{bef} \cdot C^{TH}_{bef}$. Notice that their initial count is 0 for all pairs because initially E_{aft} is empty. Since this is a data structure of the new class, it can be maintained in worst-case update time $O(m^{2/3-\varepsilon})$. In this example we know how to maintain $A^{MT} \cdot B^{TT} \cdot C^{TH}$ in worst-case update time $O(m^{2/3-\varepsilon})$ where L_1 has all medium degree vertices.

The task for $A^{MT}_{aft} \cdot B^{TT}_{bef} \cdot C^{TH}_{bef}$ is slightly different. We need to maintain this data structure for a subset of vertices in that class i.e. the vertices in L_1 transitioning from L to M (note that this is at most the number of vertices in M). Also, we need to maintain this data structure on a subset of the edges. In this example we need to use edges of $A \cap E_{aft}$, $B \cap E_{bef}$, and $C \cap E_{bef}$. But all of these edges are known so the same algorithm that we use for class M for the data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ will work for the for class of vertices transitioning from L to M for the data structure $A^{MT}_{aft} \cdot B^{TT}_{bef} \cdot C^{TH}_{bef}$.

Thus, any data structure we need to maintain with at least one of A, B, C being in E_{aft} can be maintained in worst-case update time $O(m^{2/3-\varepsilon})$. We now only worry about computing the data structures for transitioning vertices where all the edges are in E_{bef}. We will show that all such data structures can be updated in worst-case time $O(x \cdot m^{2/3-\varepsilon})$ where x is the degree of the vertex when it enters the overlapping region.

Lemma 7.7.1. *For a vertex v that is transitioning from one class to the other, the data structures for v can be computed in $O(x \cdot m^{2/3-\varepsilon})$ time, where x is the degree of v when it enters the overlapping region.*

Note that we also need to prove this for the algorithm where A and C are fixed (Section 7.3) because we use the simple algorithm as a subroutine in our main algorithm. But this is trivial because there are no edge updates in A and C so vertices in L_1 and L_4 do not change classes. Note that this algorithm only has classes for the vertices L_1 and L_4 and the classes for L_2 and L_3 are local to each chunk and are used to partition the edges of a chunk. So we do not have any vertex transitions for vertices in L_2 and L_3. Thus, we can trivially drop Assumption 4 for the algorithm where A and C are fixed. Therefore, Lemma 7.7.1 is the last piece we need to prove Theorem 14.

Proof of Theorem 14. Lemma 7.7.1 allows us to assume wlog that vertices remain in the same classes because if they change classes then we can easily update the data structures. Thus, we can now use Lemma 7.6.1, which we have already proved, to conclude the proof. ∎

7.7.2 Proof of Lemma 7.7.1

In this subsection, we only deal with matrices A_{bef}, B_{bef} and C_{bef}. We do not use A_{aft}, B_{aft} and C_{aft} at all. Thus, to simplify notation we drop the subscript "bef" and represent A_{bef}, B_{bef} and C_{bef} by A, B and C respectively.

We start with all different cases for *Type-1* transitions where the transitioning vertex is an endpoint of a path in the data structure.

Case 1: L and M
The amount of time we have in this case is $m^{1/3+\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m)$. We will only focus on vertices in L_1 transitioning from L to M because there are no data structures just for L. Note that these arguments will be similar for L_4.

Claim 7.7.2. *The amount of time it takes to compute the data structures $A^{MD} \cdot B^{DD}$ and $B^{DD} \cdot C^{DM}$ (Eq (7.14)) for a vertex $v \in L_1$ transitioning from L to M is $O(m)$.*

Proof. Consider the data structure $A^{MD} \cdot B^{DD}$ (the proof for the other one is similar). We can iterate over all neighbors of v in L_2^D and then go over all their neighbors in L_3^D to update the number of paths. This takes time $m^{1/3+\varepsilon} \cdot m^{1/3+\varepsilon} \cdot O(1)$ that is within our budget. ∎

Claim 7.7.3. *The amount of time it takes to compute the data structures $A^{MT} \cdot B^{TT} \cdot C^{TH}$ and $A^{HT} \cdot B^{TT} \cdot C^{TM}$ (Eq (7.17)) for a vertex v transitioning from L to M is $O(m)$.*

Proof. Consider the data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ for a vertex $v \in L_1$. We can iterate over all neighbors of v in L_2^T and then go over all their neighbors in L_3^T and finally over their neighbors in L_4^H to update the number of paths. This takes time $m^{1/3+\varepsilon} \cdot m^{2(1/3-2\varepsilon)} \cdot O(1) = O(m^{1-3\varepsilon})$ that is within our budget. ∎

Case 2: M and H
The amount of time we in this case have is $m^{2/3-\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{4/3-2\varepsilon})$. We focus on a vertex $v \in L_1$ transitioning from M to H or H to M.

We first consider data structures $A^{HT} \cdot B^{TS} \cdot C^{SH}$, $A^{HS} \cdot B^{ST} \cdot C^{TH}$ (Eq (7.18)).

Claim 7.7.4. *The amount of time it takes to compute the data structures $A^{HT} \cdot B^{TS} \cdot C^{SH}$ and $A^{HS} \cdot B^{ST} \cdot C^{TH}$ for a vertex $v \in L_1$ transitioning from M to H is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{HT} \cdot B^{TS} \cdot C^{SH}$ (the proof for the other is similar). We can go over all $m^{2/3-\varepsilon}$ neighbors of v in L_2^T and all their $m^{1/3-2\varepsilon}$ neighbors in L_3^S and finally all $m^{1/3+\varepsilon}$ vertices in L_4^H and update the number of paths. The time taken is $m^{2/3-\varepsilon} \cdot m^{1/3-2\varepsilon} \cdot m^{1/3+\varepsilon} = m^{4/3-2\varepsilon}$.

Consider the data structure $A^{HS} \cdot B^{ST} \cdot C^{TH}$. We can go over all $m^{2/3-\varepsilon}$ neighbors w_i of v in L_2^S, but we cannot iterate over their neighbors. So we also iterate over all $m^{1/3+\varepsilon}$ vertices u_i in L_4^H and then use the data structure $B^{*T} \cdot C^{T*}$ (Eq (7.16)) to get the number of paths between w_i and u_j through L_3^T. This takes time $m^{2/3-\varepsilon} \cdot m^{1/3+\varepsilon} \leqslant m^{4/3-2\varepsilon}$. ∎

We now consider data structures $A^{HT} \cdot B^{TT} \cdot C^{TH}$, $A^{MT} \cdot B^{TT} \cdot C^{TH}$, $A^{HT} \cdot B^{TT} \cdot C^{TM}$ (Eq (7.17)).

Claim 7.7.5. *The amount of time it takes to compute the data structures $A^{HT} \cdot B^{TT} \cdot C^{TH}$, $A^{MT} \cdot B^{TT} \cdot C^{TH}$, $A^{HT} \cdot B^{TT} \cdot C^{TM}$ for a vertex $v \in L_1$ transitioning between M and H is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ for a vertex $v \in L_1$ (the analysis for the others is the same). We can iterate over all neighbors of v in L_2^T and then go over all their neighbors in L_3^T and finally over their neighbors in L_4^H to update the number of paths. This takes time at most $2m^{2/3-\varepsilon} \cdot m^{2(1/3-2\varepsilon)} \cdot O(1) = O(m^{4/3-5\varepsilon})$ that is within our budget. ∎

We finally consider data structures of the form $A_{new}^{HS} \cdot B_{new}^{SS} \cdot C_{new}^{SH}$ (Eq (7.15)).

Claim 7.7.6. *The amount of time it takes to compute the data structures of the form $A_{old}^{HS} B_{old}^{SS} C_{new}^{SH}$ in Eq (7.15) for a vertex $v \in L_1$ transitioning from M to H is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{HS} \cdot B^{SS} \cdot C^{SH}$. We do not store this data structure, we only store this when A, B and C are restricted to some phases. But for the purpose of analysis we show how to compute the data structure for $A^{HS} \cdot B^{SS} \cdot C^{SH}$ and the analysis for all the data structures in Eq (7.15) is similar.

We can go over all $m^{2/3-\varepsilon}$ neighbors w_i of v in L_2^S, but we cannot iterate over their neighbors. So we also iterate over all $m^{1/3+\varepsilon}$ vertices u_i in L_4^H and then use the data structure $B^{*S} \cdot C^{S*}$ (Eq (7.12)) to get the number of paths between w_i and u_j through L_3^S. This takes time $m^{2/3-\varepsilon} \cdot m^{1/3+\varepsilon} \cdot O(1) \leqslant O(m^{4/3-2\varepsilon})$. ∎

Note that we do not have a case for transition between T and L because both of them do not have any data structures in L_1 (or L_4).

Case 3: S and D
The amount of time we in this case have is $m^{2/3-\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{4/3-2\varepsilon})$. We focus on a vertex $v \in L_3$ transitioning between S and D. This will be similar for vertices in L_2.

We only have to consider the data structures of the form $A^{MD}B^{DD}$ in Eq (7.14) and Eq (7.13).

Claim 7.7.7. *The amount of time it takes to compute the data structures of the form $A^{MD}B^{DD}$ in Eq (7.14) and Eq (7.13) for a vertex v transitioning from S to D is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{MD}B^{DD}$ (the analysis for the others is similar) and let $v \in L_3$. We can iterate over the $m^{2/3-\varepsilon}$ neighbors in L_1 and all $m^{1/3+\varepsilon}$ vertices in L_2^D and create the data structure in time $m^{2/3-\varepsilon} \cdot m^{1/3+\varepsilon} \cdot O(1) \leqslant O(m^{4/3-2\varepsilon})$. ∎

We now discuss all different cases for *Type-2* transitions where the transitioning vertex is an intermediate vertex of a path in the data structure.

Case 1: S and D
Consider vertices transitioning between S and D. The amount of time we have is $m^{2/3-\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{4/3-2\varepsilon})$.

We first consider the data structures of the form $A^{MD} \cdot B^{DD}$ in Eq (7.14) and Eq (7.13).

Claim 7.7.8. *The amount of time it takes to create the data structures in Eq (7.14) and Eq (7.13) for a vertex v transitioning from S to D is $O(m) \leqslant O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{MD}B^{DD}$ (the analysis for the others is similar) and let $v \in L_2$. We can iterate over the $m^{2/3-\varepsilon}$ neighbors in L_1 and all $m^{1/3+\varepsilon}$ vertices in L_3^D and create the data structure in time $m^{2/3-\varepsilon} \cdot m^{1/3+\varepsilon} \cdot O(1) = O(m) \leqslant O(m^{4/3-2\varepsilon})$. ∎

We now consider the data structures $A^{*S} \cdot B^{S*}$ and $B^{*S} \cdot C^{S*}$ (Eq (7.12)).

Claim 7.7.9. *The amount of time it takes to create the data structure $A^{*S} \cdot B^{S*}$ and $B^{*S} \cdot C^{S*}$ for a vertex v transitioning from D to S is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{*S} \cdot B^{S*}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all pairs of $2m^{2/3-\varepsilon}$ neighbors on both sides (L_1 and L_3) and create the data structure. ∎

We next consider the data structures of the form $A^{HS}_{old} B^{SS}_{old} C^{SH}_{new}$ (Eq (7.15)).

Claim 7.7.10. *The amount of time it takes to create the data structures of the form $A^{HS}_{old} B^{SS}_{old} C^{SH}_{new}$ in Eq (7.15) for a vertex $v \in L_2$ transitioning from D to S is $O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{HS} \cdot B^{SS} \cdot C^{SH}$. We do not store this data structure, we only store this when A, B and C are restricted to some phases. But for the purpose of analysis we show how to compute the data structure for $A^{HS} \cdot B^{SS} \cdot C^{SH}$ and the analysis for all the data structures in Eq (7.15) is similar. We did the same thing in the proof of Claim 7.7.6.

We go over all $2m^{2/3-\varepsilon}$ neighbors of v in L_3^S and also all pairs of high degree vertices in L_1 and L_4 and create the data structure. On the surface this seems like it takes too much time but note that all the relevant data structures of this type have either A or C in the new phase, and so we can iterate over all pairs of high degree vertices in time $O(m^{2/3-\varepsilon})$. This is similar to the proof of Claim 7.5.6 and thus, gives us the desired time bound. ∎

Finally, consider the data structures $A^{HS} \cdot B^{ST} \cdot C^{TH}$ and $A^{HT} \cdot B^{TS} \cdot C^{SH}$ (Eq (7.18)).

Claim 7.7.11. *The amount of time it takes to create the data structures $A^{HS} \cdot B^{ST} \cdot C^{TH}$ and $A^{HT} \cdot B^{TS} \cdot C^{SH}$ for a vertex v transitioning from D to S is $O(m^{2/3+2\varepsilon}) \leqslant O(m^{4/3-2\varepsilon})$.*

Proof. Consider the data structure $A^{HS} \cdot B^{ST} \cdot C^{TH}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all $m^{1/3+\varepsilon}$ neighbors of v in L_1^H and also all $m^{1/3+\varepsilon}$ high degree vertices w_i in L_4. We then use the number of paths between v and w_i through L_3^T stored in $B^{*T} \cdot C^{T*}$ (Eq (7.16)) and create the data structure. This gives us the desired time bound of $O(m^{2/3+2\varepsilon}) \leqslant O(m^{4/3-2\varepsilon})$. ∎

Case 2: T and S
Note that both T, L do not have any data structures in L_1, L_4, so we do not have to do anything for a transition between T and L. We first consider the transition from T to S. The amount of time we have is $m^{1/3-2\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{1-3\varepsilon})$.

We start with the data structures $A^{*S} \cdot B^{S*}$ and $B^{S*} \cdot C^{*S}$ (Eq (7.12)).

Claim 7.7.12. *The amount of time it takes to create the data structures $A^{*S} \cdot B^{S*}$ and $B^{S*} \cdot C^{*S}$ for a vertex v transitioning from T to S is $O(m^{2/3-4\varepsilon}) \leqslant O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{*S} \cdot B^{S*}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all pairs of $m^{1/3-2\varepsilon}$ neighbors on both sides (L_1 and L_3) and create the data structure. This takes time $m^{1/3-2\varepsilon} \cdot m^{1/3-2\varepsilon} \cdot O(1) = O(m^{2/3-4\varepsilon}) \leqslant O(m^{1-3\varepsilon})$. ∎

We now consider the data structures of the form $A^{HS}_{old} B^{SS}_{old} C^{SH}_{new}$ (Eq (7.15)).

Claim 7.7.13. *The amount of time it takes to create the data structures of the form $A^{HS}_{old} B^{SS}_{old} C^{SH}_{new}$ in Eq (7.15) for a vertex $v \in L_2$ transitioning from T to S is $O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{HS} \cdot B^{SS} \cdot C^{SH}$. We do not store this data structure, we only store this when A, B and C are restricted to some phases. But for the purpose of analysis we show how to compute the data structure for $A^{HS} \cdot B^{SS} \cdot C^{SH}$ and the analysis for all the data structures in Eq (7.15) is similar. We did the same thing in the proof of Claim 7.7.6 and Claim 7.7.10.

We go over all $m^{1/3-2\varepsilon}$ neighbors of v in L_3^S and also all pairs of high degree vertices in L_1 and L_4 and create the data structure. This seems like it takes too much time but note that all the relevant data structures of this type have either A or C in the new phase, and so we can iterate over all pairs of high degree vertices in time $O(m^{2/3-\varepsilon})$. This gives us the desired time bound of $m^{1/3-2\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{1-3\varepsilon})$. ∎

Finally, consider the data structures $A^{HS} \cdot B^{ST} \cdot C^{TH}$ and $A^{HT} \cdot B^{TS} \cdot C^{SH}$ (Eq (7.18)).

Claim 7.7.14. *The amount of time it takes to create the data structures $A^{HS} \cdot B^{ST} \cdot C^{TH}$ and $A^{HT} \cdot B^{TS} \cdot C^{SH}$ for a vertex v transitioning from T to S is $O(m^{2/3+2\varepsilon}) \leqslant O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{HS} \cdot B^{ST} \cdot C^{TH}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all $m^{1/3+\varepsilon}$ neighbors of v in L_1^H and also all $m^{1/3+\varepsilon}$ vertices w_i in L_4^H. We then use the number of paths between v and w_i through L_3^T stored in $B^{*T} \cdot C^{T*}$ (Eq (7.16)) and create the data structure. This gives us the desired time bound of $O(m^{2/3+2\varepsilon}) \leqslant O(m^{1-3\varepsilon})$. ∎

Now consider vertices transitioning from S to T. The amount of time we have is $2m^{1/3-2\varepsilon} \cdot O(m^{2/3-\varepsilon}) = O(m^{1-3\varepsilon})$. We start with the data structures $A^{*T} \cdot B^{T*}$ and $B^{T*} \cdot C^{*T}$ (Eq (7.16)).

Claim 7.7.15. *The amount of time it takes to create the data structures $A^{*T} \cdot B^{T*}$ and $B^{T*} \cdot C^{*T}$ for a vertex v transitioning from S to T is $O(m^{2/3-4\varepsilon}) \leqslant O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{*T} \cdot B^{T*}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all pairs of $2m^{1/3-2\varepsilon}$ neighbors on both sides (L_1 and L_3) and create the data structure. This takes time $2m^{1/3-2\varepsilon} \cdot 2m^{1/3-2\varepsilon} \cdot O(1) = O(m^{2/3-4\varepsilon}) \leqslant O(m^{1-3\varepsilon})$. ∎

Next, consider the data structures of the form $A^{MT} \cdot B^{TT} \cdot C^{TH}$ in Eq (7.17).

Claim 7.7.16. *The amount of time it takes to create the data structures of the form $A^{MT} \cdot B^{TT} \cdot C^{TH}$ in Eq (7.17) for a vertex v transitioning from S to T is $O(m^{2/3-\varepsilon}) \leqslant O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{MT} \cdot B^{TT} \cdot C^{TH}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all $2m^{1/3-2\varepsilon}$ neighbors of v in L_1 and also all $m^{1/3+\varepsilon}$ vertices w_i in L_4^H. We then use the number of paths between v and w_i through L_3^T stored in $B^{*T} \cdot C^{T*}$ (Eq (7.16)) and create the data structure. This gives us the desired time bound of $O(m^{2/3-\varepsilon}) \leqslant O(m^{1-3\varepsilon})$. ∎

Finally, consider the data structures $A^{HT} \cdot B^{TS} \cdot C^{SH}$ and $A^{HS} \cdot B^{ST} \cdot C^{TH}$ (Eq (7.18)).

Claim 7.7.17. *The amount of time it takes to create the data structures $A^{HT} \cdot B^{TS} \cdot C^{SH}$ and $A^{HS} \cdot B^{ST} \cdot C^{TH}$ for a vertex v transitioning from S to T is $O(m^{2/3-\varepsilon}) \leqslant O(m^{1-3\varepsilon})$.*

Proof. Consider the data structure $A^{HT} \cdot B^{TS} \cdot C^{SH}$ (the analysis for the other is similar) and let $v \in L_2$. We go over all $2m^{1/3-2\varepsilon}$ neighbors of v in L_1^H and also all $m^{1/3+\varepsilon}$ vertices w_i in L_4^H. We then use the number of paths between v and w_i through L_3^S stored in $B^{*S} \cdot C^{S*}$ (Eq (7.12)) and create the data structure. This gives us the desired time bound of $O(m^{2/3-\varepsilon}) \leqslant O(m^{1-3\varepsilon})$. ∎

7.8 Extending the Algorithm to 4-Cycles in General Graphs

In this section, we extend our algorithm for counting 4-cycles from layered graphs to general (simple) graphs.

Theorem 13. *There is an algorithm for counting the number of 4-cycles in any fully dynamic graph in worst-case update time $O(m^{2/3-\varepsilon})$ where m is the number of edges in the graph at that point and $\varepsilon > 0$ is a constant. Using the current best value of the square matrix multiplication exponent $\omega = 2.371339$ we get $\varepsilon = 0.009811$ and if we assume the best possible exponent i.e. $\omega = 2$ then we get $\varepsilon = 1/24$.*

We solve this problem by creating a layered graph G' by adding all the vertices V to each of the four layers and then running the algorithm for layered graphs that we developed in Theorem 14. The first thing we address is that when an edge update (u, v) arrives in the graph, it corresponds to 4 updates in the layered graph G'. If the update is an insertion, we insert the edge in D then C then B and then in A and if it is a deletion, we delete edges in the reverse order.

The answer to the query of the number of new 4-cycles in the general graph is the answer we get when the edge is inserted or deleted in matrix D of the layered graph. Note that even though we are counting paths in the layered graph G', these correspond to *walks* in the original graph G since vertices are repeated in all the layers, so we might count some walks that are not paths. We now show that this does not happen.

Claim 7.8.1. *The number of walks of length 3 from $u \in L_1$ to $v \in L_4$ in the layered graph is equal to the number of paths of length 3 from u to v in the general graph.*

Proof. All 3-paths in the general graph from u to v exist in the constructed layered graph, so we need to show that no extra paths exist. Consider a walk u, x, y, v of length 3 from u to v in the layered graph. We will show that all vertices on this path are distinct vertices of V so this is a path in the general graph.

Firstly we have $x \neq u$ because we do not have self loops (edges of the form (i, i)). Also, $x \neq v$ because the edge (u, v) does not exist. This is because if (u, v) is an edge insertion then we insert it in matrix D before we insert it in A, B, C so during the query this edge does not exist in A, B, C. If (u, v) is an edge deletion then we delete it from A, B, C before we query matrix D so during the query this edge does not exist in A, B, C.

The same arguments show that $y \neq u$ and $y \neq v$. Finally, we have $x \neq y$ because we do not have self loops. Thus, all the vertices are distinct and u, x, y, v is a path. ∎

Claim 7.8.1 along with Theorem 14 proves Theorem 13. Thus, the algorithm also works for general graphs and this also shows that the problem of maintaining the number of 4-cycles is equivalent in layered and general graphs.

Chapter 8

Oracle models: Learning-Augmented Maximum Independent Set

8.1 Introduction

We consider learning-augmented *maximum independent set* (MIS) in this paper. Given a (unweighted, undirected) graph $G = (V, E)$, an independent set is a set of vertices $I \subseteq V$, such that for any $u, v \in I$, $(u, v) \notin E$, i.e., there is *no* edge between u and v. The maximum independent set problem asks to find the independent set with the largest size in G.

Finding the maximum independent set is one of the classical NP-hard problems [Kar10]. Furthermore, the seminal work of [Hås96, Zuc06] demonstrates the NP-hardness of approximating the size of the MIS to within a factor of $n^{1-\delta}$ for any $\delta > 0$. In contrast, outputting any single vertex gives an n-approximation trivially. [BH92] gave a non-trivial $O(n/\log^2 n)$-approximation to MIS and this was later improved by [Fei04a]. These results indicate that the problem is quite hard in its general form and thus, many research efforts have been devoted to approximation algorithms in special settings, e.g., planar graphs [ALMM08, MM09], rectangle-intersection graphs [CC09, CE16, GKM+22], and exponential-time algorithms [Rob01, FGK06, XN13, BEPvR12].

On the other hand, heuristic algorithms, despite their bad worst-case guarantees, often exhibit commendable performance on real-world graphs [ARW12, DLS+16, WB20]. For instance, the greedy algorithm only offers an approximation guarantee of $O(\Delta)$, where Δ is the *maximum degree* of G. However, it frequently yields satisfactory empirical results. The gap between the worst-case hardness and practical efficiency motivates us to study the MIS problem through the lens of beyond worst-case analysis [Bop87, Rou21]. In particular,

under the modern context, we ask the question of finding the maximum independent set with *learning-augmented oracles*.

Learning-augmented algorithms. Learning-augmented algorithms, also known as algorithms with predictions, have attracted considerable attention in recent years (see, e.g. [PSK18,HIKV19,LMRX21,CHW22,SW23,BGTZ23,BCGL23,ACN+23,HLS+23,BFNP24, SE24], and references therein). This paradigm of beyond worst-case analysis has been successful in surmounting classical thresholds and bridging the gap between the worst-case hardness and practical efficiency (see, e.g., [MV22], for an excellent summary). Typically, in learning-augmented algorithms, we assume the access to an oracle that gives part of the "right answer" to the problem, and fails with some small but non-negligible probability. Conceptually, these algorithms aim to take advantage of modern machine learning models, which are fairly accurate on predictive tasks yet make random mistakes in an inconsistent fashion. Learning-augmented algorithms provide a great way to analyze algorithms beyond the worst case, and these algorithms usually have immediate implications in practice (see the empirical results in, e.g., [CPSZ21, HIKV19, EFS+22, SW23, ACN+23]). Inspired by the recent work in utilizing machine learning-based techniques for the maximum independent set [ASS20, PSYC21, BQS+23], we consider the MIS problem through the lens of learning-augmented algorithms.

The advantage of the learning-augmented algorithms has inspired a flurry of work that studies *graph problems* within this framework [BMS20, FS21, CSVZ22, CEI+22, HKP23, ABM23, LSV23, APT24, CdG+24, GMM24]. In a very recent work, [CdG+24] considered the Max-cut problem, in which the oracle model is closely related to our setting for the MIS problem. Under the Unique Game Conjecture (UCG), it is known that getting anything better than $\alpha \approx 0.878$ approximation for max-cut is NP-hard ([KKMO07]). In contrast, [CdG+24] showed that with a learning-augmented oracle, we could achieve better approximation than the α threshold in polynomial time. In another closely related work, [GMM24] studied the more general constraint satisfaction problems (CSPs) trough the lens of the learning-augmented algorithms. There, they obtain results for both the Max-cut and the Max 2-Lin problem. Although [CdG+24, GMM24] defines more general learning-augmented oracles, they, unfortunately, fall short of capturing the MIS-type of CSP problems, and their results do not have direct implications on the MIS problem.

From the above discussion, we can see that *a*) studying the maximum independent set problem in the framework of learning-augmented algorithms has great potential; and *b*) to this end, the existing models and algorithms are not yet sufficient. In light of this, we ask the following question:

Under the framework of learning-augmented algorithms, what efficient algorithms can we get for the maximum independent set problem?

8.1.1 Our models and contributions

In what follows, we will define the learning-augmented oracle model we consider and present our main results.

Our oracle model. We consider the following natural learning-augmented oracle: for a fixed maximum independent set I^*, the oracle answers whether a vertex $v \in I^*$ correctly with probability $1/2 + \varepsilon$, and incorrectly with probability $1/2 - \varepsilon$. In addition, the randomness is *independent* across the vertices. We denote by $\texttt{ORC}_{G,I^*}(v)$ the answer the oracle gives when queried for vertex v.

We study approximation algorithms for MIS with the learning-augmented oracle in two settings: the *persistent noise* setting and the *non-persistent noise* setting. We discuss the settings and the results, respectively, as follows.

- The **persistent noise** setting. In the persistent noise setting, the randomness of $\texttt{ORC}_{G,I^*}$ is drawn exactly *once*. Therefore, the answer for a vertex will remain the same no matter how many times you query the oracle. Another way to think about this is that the oracle can be queried at most once for a vertex. This setting is the most standard in the learning-augmented literature, and graph problems are often studied under persistent noise (see, e.g. [FS21, CSVZ22, CEI+22, XH22, HKP23, ABM23, CdG+24, GMM24] and references therein). Our main result in this setting is a randomized algorithm that with high probability[1] achieves an $\widetilde{O}(\sqrt{\Delta})$ (multiplicative) approximation to the MIS in $O(m)$ time (Theorem 15).

- The **non-persistent noise** settings. In this setting, for each vertex v, we allow $\texttt{ORC}_{G,I^*}(v)$ to use *fresh randomness* for different queries. If we are allowed to make $O(n \log n)$ queries to the oracle in total, then we can trivially recover the entire set I^* with high probability by querying each vertex $O(\log n)$ times. The interesting case is when we are allowed to make only $O(n)$ queries, i.e., a number that is *asymptotically the same* as the persistent noise setting. Although the non-persistent noise setting is less frequently studied in the learning-augmented algorithm literature, it has recently sparked considerable interest in various problems [GMT23, GSdSM24, KMBC24]. In

[1] As standard in the literature, we use "with high probability" to denote a success probability of $1 - 1/\text{poly}(n)$.

Section D.1, we show that it is easy to get an $O(\log n)$-approximation with $O(n)$ queries. Our main result considerably improved on the approximation factor: we show that we can indeed obtain an $O(1)$ approximation with $O(n)$ queries and $\widetilde{O}(m)$ runtime (Theorem 16).

Our results in the persistent noise setting hold assuming full independence, but it can be easily extended to the setting where oracle queries are assumed to use k-wise independent hash function for $k = O(\log n)$. Extending it to the pair-wise independent case is challenging as the failure probabilities in the concentration bounds are not enough for the application of a union bound.

8.1.2 Technical overview

The biggest challenge in leveraging the oracle information is distinguishing the case where $\texttt{ORC}_{G,I^*}(v)$ is indeed correct. In what follows, we give a high-level overview of our techniques describing how we can use the neighborhood information for this purpose. For the simplicity of the discussion, we always assume $\varepsilon = \Theta(1)$ in the technical overview.

Persistent noise setting: A natural approach in this setting would be to figure out the conditions in which a "yes" signal for a vertex v from the oracle implies $v \in I^*$, by aggregating signals from $N(v)$. However, such an idea is hard in the following sense. For a vertex v whose oracle query $\texttt{ORC}_{G,I^*}(v) = 1$, if there are many $u \in N(v)$ such that $\texttt{ORC}_{G,I^*}(u) = 1$, we can determine that v should *not* be in the MIS. However, the converse is not true: if a vertex v is *not* in I^*, it does *not* necessarily have many neighbors in I^*. As a result, simply aggregating neighborhood information might not be enough to determine the membership of a vertex in the MIS.

The key idea here is, instead of looking at the oracle answer for vertex v ($\texttt{ORC}_{G,I^*}(v)$), we look at what the oracle says for the *neighborhood* of the vertex v. This turns out to be a good enough signal to eliminate vertices that have *many* edges to the MIS I^*. Specifically, we can show that if v has $\widetilde{\Omega}(\sqrt{\deg(v)})$ edges to I^*, then the oracle queries for $N(v)$ contain enough information to identify such a vertex v. Upon removal of such vertices, the remaining vertices have a small degree ($\widetilde{O}(\sqrt{\Delta})$) to I^*, and a greedy independent set on the residual vertices gives a good approximation.

Non-persistent noise setting: Our algorithm for this setting is a bit more nuanced as we aim to minimize the query complexity to the oracle while aiming to achieve a good approximation. The starting point of our algorithm is from the viewpoint of the classical *pure exploration* algorithms in *multi-armed bandits (MABs)*. If we ignore the nature of

MIS in our oracle, we can reduce to the following MABs problem: given n arms with mean rewards as either $\frac{2}{3}$ or $\frac{1}{3}$, find *all* the arms with mean reward $\frac{2}{3}$ with $O(n)$ arm pulls. It is well-known that one can find a *single* best arm with high constant probability in $O(n)$ queries. The question is, can we solve the problem by resorting to purely MABs algorithms, and simply ignoring the nature of the MIS?

It turns out that the above plan is not generally feasible. In particular, we note that returning the set of *all* arms with the higher reward is very similar to finding the *top-k arms* in the MABs literature (see, e.g. [KS10, KTAS12, CLTL15, CLQ17, SJR17]). In general, it would require $\Omega(n \log k)$ arm pulls to obtain top-k arms with high constant probability ([KTAS12, SJR17]). In Section D.2, we provide lower bound results, showing that to find even $O(1)$ fraction of the high-reward arms in the instance distribution requires $\omega(n)$ queries. The lower bounds teach us that to obtain the desired query efficiency and approximation guarantee, we have to exploit the structure of the MIS.

To better understand the hardness and the insights of MABs algorithms on our problem, let us look at the elimination-based algorithm as in the classical algorithm of [EMM02, EMM06]. The first idea we can try is to adapt the elimination algorithm to our problem. To this end, a natural idea is to perform elimination based on whether the mean empirical reward of an arm is more than $\frac{1}{2}$. More concretely, we maintain a pool $\tilde{I}$ of surviving vertices and use s_r as the number of queries to each vertex in round r with $s_1 = O(1)$. In round r, we can query $\texttt{ORC}_{G,I^*}(v)$ for s_r time for each $v \in \tilde{I}$. We then eliminate all vertices $v \in \tilde{I}$ whose number of 'yes' answers is less than $s_r/2$, and recurse on the new $\tilde{I}$ to round $r+1$, for which we set $s_{r+1} = 1.5 s_r$.

Since the probability for any $v \notin I^*$ to survive decreases doubly-exponentially with the number of rounds, we can show that *all* vertices $v \notin I^*$ are eliminated after $O(\log \log n)$ rounds, and the total sample complexity on the *non-MIS* vertices is at most $O(n)$. Furthermore, the probability of losing any $v \in I^*$ decreases exponentially, we can argue that in the end, $\tilde{I}$ contains at least $\Omega(1)$ fraction of the vertices in I^*. Unfortunately, due to this fact, for each vertex $v \in I^* \cap \tilde{I}$, i.e., the vertices in the MIS that survive till the end, we need to pay for $2^{O(\log \log n)} = \operatorname{polylog} n$ on the sample complexity. Therefore, this pure exploration algorithm only works when the size of I^* is upper-bounded by $n/\operatorname{polylog} n$, and its worst-case guarantee is only a $\operatorname{polylog} n$ approximation.

Note that a $\operatorname{polylog} n$ approximation is far from what we want: after all, there is a trivial algorithm that achieves $O(\log n)$ approximation with $O(n)$ samples (see Section D.1 for details). Nevertheless, the existence of such an algorithm teaches us that the problematic case is when the MIS size is large and, in particular, *comparable* to the size of the non-MIS vertices. As such, a natural idea is to design an algorithm that handles the case when

the numbers of the MIS and the non-MIS vertices are comparable, and fuse this algorithm with the elimination-based MABs procedure we discussed above.

The above idea is quite close to the final strategy we adapt, albeit we proceed differently for the roles of the two components. In particular, we use the pure exploration MABs algorithm not to output a set with vertex set $\tilde{I} \subseteq I^*$, but to output a set of vertex set $\tilde{I}$ whose *majority (but not necessarily all) of* vertices are in I^*. To this end, we use a more *conservative* elimination strategy than the ones in the line of [EMM02, EMM06]: instead of increasing the number of samples by a multiplicative factor, we increase the number of samples in each round by an *additive* factor. In this way, we cannot guarantee that all the "wrong" arms are eliminated; however, we can argue that, since the probability for the non-MIS vertices to survive decreases exponentially, we have *i*). the number of samples used on the vertices in I^* is bounded by $O(n)$ *before* the size of $\tilde{I} \setminus I^*$ reduces to the size of $\tilde{I} \cap I^*$; and *ii*). the number of vertices in $\tilde{I} \cap I^*$ only decreases by a constant fraction. In this way, we can design an efficient procedure that eliminates the "surplus" non-MIS vertices to always create cases when the number of non-MIS vertices is smaller.

The final missing piece is the MIS algorithm that deals with the case when the number of MIS vertices takes the majority of the vertex set. Our algorithm to handle this case is to compute an *approximate vertex cover* of the graph and the remaining vertices will form an approximate independent set. It is a well-known fact that if we compute a *maximal matching* and take *all* their endpoints, it forms a 2-approximate vertex cover that covers all edges in the graph. Furthermore, since the size of the non-MIS vertices is small, there can be only a *limited number* of vertices $v \in I^*$ that can be counted in the vertex cover. As such, we can simply *remove* these vertices from the graph. The rest of the graph would form an independent set, and since we remove at most a constant fraction of vertices from I^* throughout the two phases, we get an $O(1)$ approximation.

Notation. We let I^* denote a fixed maximum independent set of the graph G. We let $N_{I^*}(v) = N(v) \cap I^*$ be the set of neighbors of the vertex v in the independent set and let $\deg_{I^*}(v) := |N_{I^*}(v)|$ be its size. Furthermore, we let $\widetilde{N_{I^*}}(v)$ be the set of neighbors of the vertex v that are claimed to be in the independent set by the oracle and let $\widetilde{\deg}_{I^*}(v)$ be its size.

8.2 An Algorithm in the Persistent Noise Setting

In this section we present an algorithm for the learning-augmented MIS problem with persistent noise. Formally we prove the following

Theorem 15. *There exists a randomized algorithm that given*

a) an input graph $G = (V, E)$ with maximum degree Δ and

b) an MIS oracle $\texttt{ORC}_{G,I^*}$ with persistent noise *for an unknown maximum independent set I^*,*

in $O(m)$ time outputs an independent set I such that $|I| \geqslant \frac{\varepsilon}{12} \cdot (\Delta \ln n)^{-0.5} \cdot |I^*|$ with high probability.

We dedicate the remainder of this section to the proof of Theorem 15. We start with the assumption that $\varepsilon \leqslant 1/4$ (we can do this for any constant > 0). This assumption is needed for technical reasons. If $\varepsilon > 1/4$, then it is easy to simulate an oracle with $\varepsilon = 1/4$ by flipping the oracle answer with probability $p = \frac{\varepsilon - 1/4}{1/2+\varepsilon}$ ($p \geqslant 0$ since $\varepsilon > 1/4$). If we do this then the probability that the oracle gives the incorrect answer is $(1/2-\varepsilon)+p\cdot(1/2+\varepsilon) = 1/4$ which is exactly what we wanted. Note that the final bound we get on the approximation factor now changes by a factor of at most 2. This is because when $\varepsilon > 1/4$ we are replacing it with an oracle for $\varepsilon = 1/4$ and the approximation factor linearly depends on ε.

The algorithm and analysis. We now state our algorithm.

Algorithm 15. An algorithm for MIS in persistent noise setting.
Input: A graph $G = (V, E)$ with maximum degree Δ that contains an unknown maximum independent set I^*; an MIS oracle $\texttt{ORC}_{G,I^*}$ in the persistent noise setting
Output: A set of vertices I such that I forms an independent set and $|I| \geqslant \frac{\varepsilon}{3} \cdot (\Delta \ln n)^{-0.5} \cdot |I^*|$.
Parameters: $s_v := (1/2 - \varepsilon)\deg(v) + 6\sqrt{\ln n} \cdot (1/2 - \varepsilon)\sqrt{\deg(v)}$.

1. Calculate $\widetilde{\deg}_{I^*}(v)$ for all vertices $v \in V$.
2. Let L be the set of vertices where $\deg(v) \leqslant 36 \ln n$ for $v \in V$.
3. Let S be the set of vertices where $\widetilde{\deg}_{I^*}(v) \leqslant s_v$ for $v \in V \setminus L$.
4. Output the greedy MIS I on $G[S \cup L]$.

We first show that if $v \in I^*$, the number of "yes" answers in $N(v)$ cannot be too high.

Claim 8.2.1. *If $v \in I^* \setminus L$ then with high probability, $\widetilde{\deg}_{I^*}(v) \leqslant (1/2-\varepsilon)\deg(v)+6\sqrt{\ln n}\cdot (1/2 - \varepsilon)\sqrt{\deg(v)}$.*

Proof. If $v \in I^*$ then $\deg_{I^*}(v) = 0$ which means that the expected size of $\widetilde{\deg}_{I^*}(v)$ is $(1/2 - \varepsilon)\deg(v)$. Since we assume complete independence for the oracle we can use the Chernoff bound to get concentration.

Let $X_i = 1$ if i^{th} neighbor is claimed to be in I^* by the oracle where $i \in [\deg(v)]$. Observe that $\widetilde{\deg}_{I^*}(v) = \sum_i X_i$ is the number of neighbors that claim to be in I^*. We know $\mu = \mathbb{E}\left[\widetilde{\deg}_{I^*}(v)\right] = (1/2 - \varepsilon)\deg(v)$. Using the Chernoff (Proposition 2.4.3) bound with $\delta_v = 6\left(\frac{\ln n}{\deg(v)}\right)^{0.5} \leqslant 1$:

$$\Pr\left(\widetilde{\deg}_{I^*}(v) > (1+\delta_v)\mu\right) \leqslant \exp\left(-\frac{\delta_v^2 \cdot \mu}{3}\right) \leqslant n^{-3}. \qquad \text{(since } \varepsilon \leqslant 1/4\text{)}$$

Notice that as $\deg(v)$ gets larger we get better concentration. ∎

Note that Claim 8.2.1 does *not* rule out the case that a vertex $v \in V \setminus I^*$ and has very few neighbors in I^*. Nevertheless, it tells us that if we simply eliminate the vertices that "block" a large number of neighbors in I^*, we will not mistakenly drop vertices in I^*.

Next, we show that if a vertex v has many neighbors in I^* i.e. $\deg_{I^*}(v)$ is large then $\widetilde{\deg}_{I^*}(v)$ should also be large and hence we should be able to detect such a vertex $v \notin I^*$.

Claim 8.2.2. *If $v \notin I^*$ and $\deg_{I^*}(v) \geqslant (3/\varepsilon)\sqrt{\ln n}\sqrt{\deg(v)}$ then with high probability, $\widetilde{\deg}_{I^*}(v) > (1/2 - \varepsilon)\deg(v) + 6\sqrt{\ln n} \cdot (1/2 - \varepsilon)\sqrt{\deg(v)}$.*

Proof. If $v \notin I^*$ and $\deg_{I^*}(v) = k$ then the expected size of $\widetilde{\deg}_{I^*}(v)$ is

$$\mu = \mathbb{E}\left[\widetilde{\deg}_{I^*}(v)\right] = k(1/2 + \varepsilon) + (\deg(v) - k)(1/2 - \varepsilon) = (1/2 - \varepsilon)\deg(v) + 2\varepsilon k.$$

We now use the Chernoff bound (Proposition 2.4.3) with $t = \varepsilon k$ to get concentration:

$$\begin{aligned}\Pr\left(\widetilde{\deg}_{I^*}(v) < \mu - t\right) &\leqslant \exp\left(-2t^2/\deg(v)\right) \\ &= \exp\left(-2\varepsilon^2 k^2/\deg(v)\right) \\ &\leqslant n^{-3}. \qquad \text{(using the lower bound on k)}\end{aligned}$$

Thus, with high probability we have:

$$\widetilde{\deg}_{I^*}(v) \geqslant \mu - \varepsilon k$$

$$
\begin{aligned}
&= (1/2-\varepsilon)\deg(v) + \varepsilon k \\
&= (1/2-\varepsilon)\deg(v) + 3\sqrt{\ln n}\sqrt{\deg(v)} \\
&> (1/2-\varepsilon)\deg(v) + 6\sqrt{\ln n}\cdot(1/2-\varepsilon)\sqrt{\deg(v)}\,.
\end{aligned}
$$

∎

We can conclude that the events in Claim 8.2.1 and Claim 8.2.2 happen with high probability by a union bound over all vertices.

Finalizing the proof of Theorem 15. Calculating $\widetilde{\deg}_{I^*}(v)$ for all vertices $v \in V$ and finding set S takes $O(m)$ time. The greedy MIS can also be computed in $O(m)$ time.

We first condition on the events in Claim 8.2.1 and Claim 8.2.2 for all vertices (this happens with high probability). Notice that for all vertices in $v \in S$ we have $\widetilde{\deg}_{I^*}(v) \leqslant s_v$. By Claim 8.2.1 all vertices in I^* are in S. By Claim 8.2.2 we know that any non-MIS vertices v that are in S have $\deg_{I^*}(v) \leqslant (3/\varepsilon)\sqrt{\ln n}\sqrt{\deg(v)} \leqslant (6/\varepsilon)\sqrt{\Delta \ln n}$. Also, vertices in L have $\deg_{I^*}(v) \leqslant \deg(v) = \sqrt{\deg(v)}\cdot\sqrt{\deg(v)} \leqslant \sqrt{\Delta}\sqrt{36\ln n} \leqslant (6/\varepsilon)\sqrt{\Delta \ln n}$.

This means that when we run the greedy MIS algorithm and pick a non-MIS vertex, we eliminate at most $(6/\varepsilon)\sqrt{\Delta \ln n}$ vertices in I^*. Thus, we have $|I| \geqslant \frac{\varepsilon}{6}\cdot(\Delta \ln n)^{-0.5}\cdot|I^*|$. Finally, because of the assumption on ε ($\varepsilon \leqslant 1/4$), we lose a factor of at most 2 in the approximation, giving us the final bound $|I| \geqslant \frac{\varepsilon}{12}\cdot(\Delta \ln n)^{-0.5}\cdot|I^*|$. ∎

Remark 8.2.3. We assume that we have complete independence between the oracle queries for the vertices. But we can get essentially the same result (up to constants) when the oracle answers the queries using a k-wise independent hash function instead of a completely random function for $k = O(\log n)$.

This holds because we use Proposition 2.4.8 with $k = O(\log n)$ instead of the Chernoff bound (Proposition 2.4.3). The min in the exponent always picks the second term because k is large enough and so we get something very similar to the Chernoff bound in Proposition 2.4.3 where the exponent only differs by some constants. Thus, the approximation we get will be a small constant factor worse but will remain the same asymptotically.

8.3 An Algorithm in the Non-persistent Noise Setting

In this section, we consider algorithms in the *non-persistent noise setting (MABs setting)* of the MIS oracle, i.e., the algorithm can access the learning-augmented MIS oracle with *fresh randomness* for each query of a vertex v. The formal statement of our main result in this setting is as follows.

Theorem 16. *There exists a randomized algorithm that given a parameter $\delta \in (0,1)$ and*

a) an input graph $G = (V, E)$ with a maximum independent set I^*; and

b) an MIS oracle $\mathtt{ORC}_{G,I^*}$ in the non-persistent noise setting*,*

with probability at least $(1-\delta)$, in $O(m \log n)$ time and $\frac{30n}{\varepsilon^2} \cdot \log \frac{1}{\delta}$ total queries to $\mathtt{ORC}_{G,I^*}$, computes a set I such that $|I| \geqslant \frac{48}{50} \cdot |I^*|$.

We dedicate the remainder of this section to the proof of Theorem 16.

The algorithm. As we have discussed in our high-level overview, our algorithm proceeds in two phases. In the first phase, our algorithm focuses on eliminating most of the vertices in the non-MIS vertex set. Then, in the second phase, we show that a good approximation to vertex cover is enough to get a good approximation to the independent set. We can easily find a 2-approximate vertex cover in $O(m)$ time by computing a maximal matching and picking all its endpoints. The detailed description of the algorithm is as follows.

Algorithm 16. An algorithm for MIS in non-persistent noise setting.
Input: A graph $G = (V, E)$ that contains an unknown maximum independent set I^*; an MIS oracle $\mathtt{ORC}_{G,I^*}$ in the multi-armed bandit setting; a confidence parameter $\delta \in (0,1)$.
Output: A set of vertices I such that I forms an independent set and $|I| = O(|I^*|)$.
Parameters: $q_r = \frac{4}{\varepsilon^2} \cdot (r + \log \frac{1}{\delta})$.

- Maintain a set of V_r with the initialization $V_0 \leftarrow V$.
- For $r = 1$ to ∞, do the following:
  1. **Elimination phase:**
     - For each vertex $v \in V_{r-1}$:
       (a) Query v for q_r times.
       (b) *Remove* v from V_{r-1} if the number of 1 returned by $\mathtt{ORC}_{G,I^*}(v)$ ("yes" answers) is less than $q_r/2$.
     - Let the updated vertex set be V_r, i.e., V_r is a subset of vertices of V_{r-1} that gets at least $q_r/2$ "yes" answers from $\mathtt{ORC}_{G,I^*}$ (v).
  2. **Vertex Cover phase:**

(a) Compute a 2-approximate vertex cover U_r of the induced subgraph $G[V_r]$.

(b) Let $I_r \leftarrow V_r \setminus U_r$.

3. Maintain the set I with the maximum size among all I_r's, i.e., let $I \leftarrow I_r$ if I_r is larger than I and keep I unchanged otherwise.
4. If the total number of $\texttt{ORC}_{G,I^*}$ queries is more than $30 \cdot \frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta}$ then terminate and return the currently maintained I.

Note that since we do *not* necessarily know the actual size of I^*, we compute a vertex cover after every elimination phase and simply output the independent set with the largest size throughout the process.

The analysis. We now proceed to the analysis of the algorithm. Before diving into the main lemmas, we first show some straightforward technical claims that characterize the behavior of the MIS and non-MIS vertices in the elimination phase. We first show that the probabilities of an MIS vertex being eliminated and a non-MIS vertex surviving in round r are both small.

Claim 8.3.1. *The following statements are true:*

1. *Let $v \in V_{r-1} \cap I^*$; then, the probability that v is removed from V_r is at most $\frac{1}{100} \cdot \frac{\delta}{4^r}$.*
2. *Let $v \in V_{r-1} \setminus I^*$; then, the probability that v is* not *removed from V_r is at most $\frac{1}{100} \cdot \frac{\delta}{4^r}$.*

Proof. We prove this claim by applying the Chernoff bound in Proposition 2.4.3. For any vertex $v \in I^*$, let the random variable $X_v^i = 1$ if the i^{th} query for vertex v is a "yes" and $X_v^i = 0$ otherwise for $i \in [q_r]$. Observe that $X_v = \sum_i X_v^i$ is the number of "yes" answers returned by $\texttt{ORC}_{G,I^*}(v)$ out of the q_r queries. Clearly, we have that $\mathbb{E}[X_v] = (1/2 + \varepsilon) \cdot q_r$, and X_v is a summation of the independent indicator random variables so, we can apply Proposition 2.4.3 to show that

$$\begin{aligned}
\Pr\left(X_v < \frac{q_r}{2}\right) &= \Pr(X_v - \mathbb{E}[X_v] \leqslant -\varepsilon \cdot q_r) \\
&\leqslant \exp\left(-2 \cdot \varepsilon^2 \cdot q_r\right) && \text{(applying Proposition 2.4.3)} \\
&= \exp\left(-8r - 8 \log \frac{1}{\delta}\right) && \text{(by the definition of } q_r\text{)}
\end{aligned}$$

$$\leqslant \exp(-6) \cdot \exp(-2r) \cdot \exp\left(-8 \log \frac{1}{\delta}\right)$$
$$\leqslant \frac{1}{100} \cdot \frac{\delta}{4^r}.$$

Note that the vertices in $v \in I^* \cap V_{r-1}$ are in I^*. Therefore, we can get the desired statement for $v \in I^* \cap V_{r-1}$.

We can similarly define Y_v for the number of "yes" answers returned by $\texttt{ORC}_{G,I^*}(v)$ with q_r queries for a vertex $v \in V \setminus I^*$. Here, we have that $\mathbb{E}[Y_v] = (1/2 - \varepsilon)q_r$. As such, we have that

$$\begin{aligned}
\Pr\left(Y_v \geqslant \frac{q_r}{2}\right) &= \Pr(Y_v - \mathbb{E}[Y_v] \geqslant \varepsilon \cdot q_r) \\
&\leqslant \exp\left(-2 \cdot \varepsilon^2 \cdot q_r\right) && \text{(applying Proposition 2.4.3)} \\
&= \exp\left(-8r - 8 \log \frac{1}{\delta}\right) && \text{(by the definition of } q_r\text{)} \\
&\leqslant \exp(-6) \cdot \exp(-2r) \cdot \exp\left(-8 \log \frac{1}{\delta}\right) \\
&\leqslant \frac{1}{100} \cdot \frac{\delta}{4^r}.
\end{aligned}$$

This gives us the desired statement for $v \in V_{r-1} \setminus I^*$ as well. Claim 8.3.1 □

We now prove the main technical lemma of our algorithm that helps eventually prove Theorem 16. In what follows, we will denote the size of I^* as αn for some $\alpha \in (0, 1)$. Our main lemma for the elimination phase is as follows.

Lemma 8.3.2. *Let $|I^*| = \alpha n$ for some $\alpha \in (0, 1)$ and $\tilde{r} = 1 + \log \frac{1}{\alpha}$. With probability at least $1 - \delta$, the following statements about Algorithm 16 are true:*

i). The number of vertices in $V_{\tilde{r}}$ that are not *in I^* is at most $\alpha n/100$, i,e,*

$$|V_{\tilde{r}} \setminus I^*| \leqslant \frac{\alpha n}{100}.$$

ii). The number of vertices in $V_{\tilde{r}}$ that are in I^* is at least $49/50 \cdot \alpha n$, i.e.,

$$|V_{\tilde{r}} \cap I^*| \geqslant \frac{49}{50} \cdot \alpha n.$$

iii). The total *number of* $\texttt{ORC}_{G,I^*}$ *queries in the first* $\tilde{r}$ *rounds is at most* $30n/\varepsilon^2 \cdot \log 1/\delta$*, i.e.,*

$$\sum_{r=1}^{\tilde{r}} |V_{r-1}| \cdot q_r \leqslant 30 \cdot \frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta}.$$

Note that in the above, $|V_{r-1}| \cdot q_r$ *is exactly the number of queries used in round* r*.*

Proof. We prove the statements in order.

Proof of *i*)**.** Note that by Claim 8.3.1, the probability that a vertex in $V \setminus I^*$ survives round r is at most $\frac{1}{100} \cdot \frac{\delta}{4^r}$. As such, we have that

$$\begin{aligned}
\mathbb{E}\left[|V_{\tilde{r}} \setminus I^*|\right] &= \sum_{v \in V_{\tilde{r}-1} \setminus I^*} \Pr(v \text{ survives round } \tilde{r}) \\
&= \sum_{v \in V \setminus I^*} \Pr(v \text{ survives all rounds till } \tilde{r}) \\
&= \sum_{v \in V \setminus I^*} \prod_{i=1}^{\tilde{r}} \Pr(v \text{ survives round } i \mid v \text{ survives all rounds till } i-1) \\
&\qquad\qquad \text{(All rounds are independent)} \\
&\leqslant \sum_{v \in V \setminus I^*} \prod_{i=1}^{\tilde{r}} \frac{\delta}{100} \cdot \frac{1}{4^i} \\
&\leqslant n \cdot \left(\frac{\delta}{100}\right)^{\tilde{r}} \cdot \left(\frac{1}{4}\right)^{\binom{\tilde{r}}{2}} \\
&\leqslant \frac{\delta n}{100} \cdot \left(\frac{1}{4}\right)^{\tilde{r}} \\
&\leqslant \frac{\alpha \cdot n \cdot \delta}{400}. \qquad \text{(using } \alpha \in (0,1)\text{)}
\end{aligned}$$

Therefore, by Markov inequality, we have

$$\Pr\left(|V_r \setminus I^*| > \frac{\alpha n}{100}\right) \leqslant \frac{\delta}{4}$$

as desired.

Proof of *ii*)**.** By Claim 8.3.1, the probability that a vertex v is eliminated in round r is at most $\frac{\delta}{100} \cdot \frac{1}{4^r}$. We analyze the number of vertices in I^* that are eliminated by round r. We can show that the expected value is

$$\begin{aligned}
\mathbb{E}\left[|I^* \setminus V_{\tilde{r}}|\right] &= \sum_{v \in I^*} \Pr(v \text{ is eliminated by round } \tilde{r}) \\
&\leqslant \sum_{v \in I^*} \sum_{i=1}^{\tilde{r}} \Pr(v \text{ is eliminated in round } i) && \text{(Union Bound)} \\
&\leqslant \sum_{v \in I^*} \sum_{i=1}^{\tilde{r}} \frac{\delta}{100} \cdot \frac{1}{4^i} \\
&\leqslant (\alpha n) \cdot \frac{\delta}{100} \cdot \frac{1}{3}. && \text{(Geometric Sum)}
\end{aligned}$$

Therefore, by a simple Markov bound, we have that

$$\Pr\left(|I^* \setminus V_r| > \frac{\alpha n}{50}\right) \leqslant \frac{\delta}{6}.$$

Thus, with probability at least $1 - \delta/6$ we have $|I^* \cap V_{\tilde{r}}| \geqslant \frac{49}{50} \cdot \alpha n$.

Proof of *iii*)**.** Note that we are proving this bound holds even if we remove the termination condition from the algorithm. This will show that we will reach round $\tilde{r}$ with high probability. We first condition on the events in the proofs of i) and ii). Note that, unlike the standard analysis of elimination-based algorithms, here, we cannot directly upper-bound the total number of queries each round. Instead, we separately analyze the number of queries induced by the vertices in I^* and $V \setminus I^*$.

We first analyze the number of queries induced by the vertices in $V \setminus I^*$. Let us define $X_{\neg I^*}$ as the total number of queries induced by the non-MIS vertices. Similarly, we can define $X^r_{\neg I^*}$ as the queries induced by the non-MIS vertices at round r. Thus, we have that

$$\begin{aligned}
\mathbb{E}\left[X_{\neg I^*}\right] &= \sum_{v \in V - I^*} \sum_{i=1}^{\tilde{r}} \Pr(v \text{ survives till round } i) \cdot q_i \\
&\leqslant n \sum_{i=1}^{\tilde{r}} q_i \prod_{j=1}^{i} \Pr(v \text{ survives round } j \mid v \text{ survives till round } j-1) \\
&\leqslant n \sum_{i=1}^{\tilde{r}} q_i \prod_{j=1}^{i} \frac{\delta}{100} \cdot \frac{1}{4^j} && \text{(Claim 8.3.1)}
\end{aligned}$$

$$
\begin{aligned}
&\leqslant n \sum_{i=1}^{\tilde{r}} \left(\frac{\delta}{100}\right)^i \cdot \left(\frac{1}{4}\right)^{\binom{i}{2}} \cdot q_i \\
&= n \sum_{i=1}^{\tilde{r}} \left(\frac{\delta}{100}\right)^i \cdot \left(\frac{1}{4}\right)^{\binom{i}{2}} \cdot \frac{4}{\varepsilon^2} \cdot (i + \log 1/\delta) \\
&\leqslant \frac{4\delta n}{100\varepsilon^2} \sum_{i=1}^{\tilde{r}} \left(\frac{1}{4}\right)^i \cdot (i + \log 1/\delta) && \text{(Since } \delta \leqslant 1\text{)} \\
&\leqslant \frac{4\delta n}{100\varepsilon^2}(1 + \log 1/\delta). && \text{(using properties of geometric sums)}
\end{aligned}
$$

Therefore, by Markov inequality, we can show that

$$
\Pr\left(X_{\neg I^*} > \frac{2n}{5\varepsilon^2} \log 1/\delta\right) \leqslant \delta/5.
$$

We now analyze the queries induced by the vertices in I^*. Similar to the case of the non-MIS analysis, let us define X_{I^*} as the total number of queries induced by the MIS vertices. We will trivially upper bound X_{I^*} in the following way:

$$
\begin{aligned}
X_{I^*} &\leqslant \alpha n \sum_{i=1}^{\tilde{r}} q_i \\
&= \alpha n \sum_{i=1}^{\tilde{r}} \frac{4}{\varepsilon^2} \cdot \left(i + \log \frac{1}{\delta}\right) \\
&\leqslant \frac{4\alpha n}{\varepsilon^2} \cdot \left(\tilde{r}^2 + \tilde{r} \cdot \log \frac{1}{\delta}\right) \\
&\leqslant \frac{4\alpha n}{\varepsilon^2} \cdot \left(1 + (\log 1/\alpha)^2 + \lg 1/\alpha \cdot (2 + \log 1/\delta) + \log 1/\delta\right) \\
&\leqslant \frac{4n}{\varepsilon^2} \cdot (5 + 2\log 1/\delta) && \text{(using } \alpha \cdot \lg \tfrac{1}{\alpha} \leqslant 1 \text{ and } \alpha \cdot \lg^2 \tfrac{1}{\alpha} \leqslant 2 \text{ for any } \alpha \in (0,1)\text{)}
\end{aligned}
$$

We can then add the number of queries used by $X_{\neg I^*}$ and X_{I^*} to get the desired sample complexity bound of $\frac{30n}{\varepsilon^2} \cdot \log 1/\delta$.

Finally, we can apply a union bound over the failure probabilities of the events in the proofs of *i*), *ii*), and *ii*) to argue that with probability at least $1 - \delta$, all the statements hold. Lemma 8.3.2 □

We now proceed to show the guarantee of the matching and MIS phase. Our main lemma for this part is as follows.

Lemma 8.3.3. *Let $V_r \subseteq V$ be any subset of vertices in Algorithm 16. Furthermore, assume that the number of MIS vertices in V_r is at least 50 times the number of non-MIS vertices in V_r, i.e.,*

$$|V_r \cap I^*| \geqslant 50 \cdot |V_r \setminus I^*| .$$

Then, the set I_r returned by Algorithm 16 is a valid independent set, and we have

$$|I_r| \geqslant \frac{49}{50} \cdot |V_r \cap I^*| .$$

Proof. Recall that we compute a 2-approximate vertex cover U_r in the vertex cover phase. We know that the complement $I_r \leftarrow V_r \setminus U_r$ is an independent set. This is because all edges of the graph are incident on the vertex cover so the remaining vertices form an independent set.

We know that $V_r \setminus I^*$ is a vertex cover since $V_r \cap I^*$ is an independent set. Thus, we have

$$\begin{aligned}
|I_r| &= |V_r| - |U_r| && \text{(by definition)} \\
&\geqslant |V_r \cap I^*| + |V_r \setminus I^*| - 2\,|V_r \setminus I^*| && \text{(since } U_r \text{ is a 2-approximation)} \\
&\geqslant |V_r \cap I^*| - \frac{1}{50} \cdot |V_r \cap I^*| && \text{(using the assumption)} \\
&= \frac{49}{50} \cdot |V_r \cap I^*|
\end{aligned}$$

Lemma 8.3.3 □

The final missing piece is the *efficiency* of the algorithm. We now prove that the algorithm is efficient both in time and the number of $\texttt{ORC}_{G,I^*}$ oracle queries.

Lemma 8.3.4. *Algorithm 16 runs in $O(m \log n)$ time and uses at most $\frac{30n}{\varepsilon^2} \cdot \log \frac{1}{\delta}$ queries on $\texttt{ORC}_{G,I^*}$.*

Proof. The query complexity is by the design of the algorithm as we terminate upon using more than $30 \cdot \frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta}$ queries.

For the running time, note that in each iteration of r, we only need to: *i*). take the majority for all queried vertices, which can be maintained in $O(n)$ time; and *ii*). compute

a greedy matching and remove the vertices, which takes $O(m)$ time. By Lemma 8.3.2, the process terminates in $O(\log \frac{1}{\alpha}) = O(\log n)$ time ($\alpha \geqslant \frac{1}{n}$ since there has to be at least one vertex in I^*). Therefore, the entire algorithm takes $O(m \log n)$ time in total. ∎

Finalizing the proof of Theorem 16. The query efficiency is by the design of the algorithm, and the running time simply follows from Lemma 8.3.4. For the approximation guarantee, note that by Lemma 8.3.2, we will proceed to round $\tilde{r} = 10 \log \frac{1}{\alpha}$, at which point we will have $|V_{\tilde{r}} \cap I^*| \geqslant \frac{49}{50} \cdot \alpha n$ and $|V_r \cap I^*| \geqslant 50 \cdot |V_r \setminus I^*|$. Therefore, by Lemma 8.3.3, the returned $I_{\tilde{r}}$ is of size at least

$$|I_{\tilde{r}}| \geqslant \frac{49}{50} \cdot |V_{\tilde{r}} \cap I^*| \geqslant \frac{49}{50} \cdot \frac{49}{50} \cdot \alpha n,$$

which gives us the desired 48/50 approximation. ∎

Remark 8.3.5. We aim to get the $O(1)$ approximation in our algorithm and analysis. However, we remark that we can get both non-asymptotic and asymptotic trade-offs between the number of queries and the approximation factor. For the non-asymptotic trade-off (i.e., using more queries to get a better constant approximation), we can increase the leading constant on the sample complexity, and obtain the approximation with a larger constant. For the asymptotic trade-off, we can perform the simple trick by sampling k vertices uniformly at random and running Algorithm 16 on the sampled vertices. This will give us an $O(\frac{k}{n})$-approximation algorithm with $O(\frac{k}{\varepsilon^2} \cdot \log \frac{1}{\delta})$ queries as long as $\alpha k = \Omega(\log n)$.

Chapter 9

Oracle Models: Streaming Verification

9.1 Introduction

In the classical two-player communication model, the players Alice and Bob each receive an input unknown to the other player and exchange as few bits as possible to compute a function of the inputs. In their seminal paper on communication complexity classes, Babai, Frankl, and Simon [BFS86] defined the Merlin-Arthur (MA) communication model, where there is also a "super-player" Merlin, who knows the inputs of both Alice and Bob (collectively "Arthur") and hence, also knows the output. However, Merlin is not trusted, and so he provides Arthur a "proof" that should be thought of as a help message. After receiving this message, Alice and Bob communicate between themselves to verify its correctness. It turns out that for most well-studied functions, the total communication needed between the players can be significantly reduced with the help of Merlin. This holds even in the most restrictive version of the MA model, where both Merlin and Alice send just a single message to Bob, from which he must verify the solution. This is known as the *online Merlin-Arthur* (OMA) model. We focus on this setting and prove new lower bounds that also extend to a related stream-verification model called *annotated streaming* [CCM09, CCMT14]. We discuss the motivation and context of our results below.

9.1.1 Motivation and Context

Online Merlin-Arthur communication. The OMA complexity of a function is defined as the total communication between the players, i.e., the sum of the lengths of Merlin's

and Alice's messages, in the optimal protocol that minimizes this sum. In a protocol, we denote Merlin's message length by hcost (shorthand for *help cost*) and Alice's message length by vcost (shorthand for *verification cost*), while the sum of their message lengths is termed tcost (short for *total cost*). Thus, the OMA complexity of a function is the minimum tcost over all possible OMA protocols computing the function. Naturally, a function with high OMA complexity, possibly close to the input size N, is deemed "hard" in the OMA model. Over the years, researchers have wondered what problems are hard in this model [BFS86,AW08,CCGT14,Gav21]. Surprisingly, Aaronson and Wigderson [AW08] showed that even the "hardest" functions in classical communication complexity, namely the disjointness and inner product functions, have OMA complexity only $O(\sqrt{N}\log N)$. Although the same authors showed (via a simple counting argument) the existence of functions with OMA complexity $\Omega(N)$, exhibiting an *explicit* function even with OMA complexity $\omega(\sqrt{N})$ remains a longstanding open problem.

Another notion of "hard functions" in this setting might be ones that need large hcost in order to reduce the vcost from what Alice would have already needed to send without Merlin (i.e., in the classical one-way communication model). This implies high OMA complexity of these functions *relative* to their classical one-way complexity (which might be much smaller than the input size, rendering them *not* hard in the earlier sense). Again, disjointness or inner product are not hard in this sense either: Aaronson and Wigderson's protocol shows that for these problems, hcost and vcost can be smoothly traded off while keeping the product $\tilde{\Theta}(N)$. This means even a proof of size slightly larger than $\log N$ suffices to reduce Alice's message size to $o(N)$, whereas she would need to communicate $\Omega(N)$ in the classical setting. Further, prior work has shown that for many other hard functions in classical communication complexity, we can achieve the tradeoff $\mathsf{hcost}\cdot\mathsf{vcost} = O(N)$ [CCM09, CCGT14, CGT20], ruling them out from being hard in the OMA model. Interestingly, there is a complementary general lower bound that says that an OMA protocol solving *any* function f must have $\mathsf{hcost}\cdot\mathsf{vcost} = \Omega(R^{\rightarrow}(f))$ [BFS86], where $R^{\rightarrow}(f)$ is the one-way communication complexity of f. Observe that this lower bound implies that the OMA complexity $\mathrm{MA}^{\rightarrow}(f)$ of the function f must be at least $\Omega(\sqrt{R^{\rightarrow}(f)})$. Stronger lower bounds are hardly known, with a couple of exceptions.[1]

Prior work has shown stronger lower bounds for the *sparse* version of the fundamental index problem [CCGT14], and testing connectivity and bipartiteness of the *XOR* of two graphs [Tha16b]. All these functions have OMA complexity linear (or nearly linear) in their classical one-way complexity. This means the *trivial* OMA protocol, where Merlin sends an empty string (or a junk message) while Alice and Bob (optimally) solve the problem

[1]Indeed, these exceptions have $R^{\rightarrow}(f) = o(N)$ and they do not break the $\sqrt{N}$-barrier for OMA complexity.

on their own, is (nearly) as good as any other protocol for the problem. Further, since the lower bounds imply that an OMA protocol for any of these functions f must have tcost $\tilde{\Omega}(R(f))$, it follows that reducing the vcost to $o(R^{\rightarrow}(f))$ necessitates hcost to be $\tilde{\Omega}(R^{\rightarrow}(f))$. Hence, these problems are hard in the second sense described above. Given this ray of hope on identifying hard OMA problems, we pursue this thread. In this work, we define the notion of *non-trivial* OMA complexity of a function f as follows. Call an OMA protocol computing f as *non-trivial* if its vcost is $o(R^{\rightarrow}(f))$. Then, the non-trivial OMA complexity $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ is the minimum tcost over all non-trivial protocols for f.

Observe that for functions f with standard OMA complexity $\mathrm{MA}^{\rightarrow}(f) = o(R^{\rightarrow}(f))$, we have $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \mathrm{MA}^{\rightarrow}(f)$ (since the protocol minimizing tcost must be non-trivial); whereas if $\mathrm{MA}^{\rightarrow}(f) = \Omega(R^{\rightarrow}(f))$, then hcost dominates vcost in any non-trivial protocol for f, and $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ essentially measures the minimum hcost required to achieve $\mathsf{vcost} = o(R^{\rightarrow}(f))$. Thus, non-trivial OMA complexity formally captures the "relative" notion of hardness discussed above. Notably, in terms of the input size N, the best-known lower bound on $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ was $\tilde{\Omega}(\sqrt{N})$ (since a bound better than $\tilde{\Omega}(\mathrm{MA}^{\rightarrow}(f))$ was unknown, and $\mathrm{MA}^{\rightarrow}(f)$ has a longstanding $\sqrt{N}$-barrier).

It is now natural to ask the following questions.

- *Can we find a single problem with high non-trivial OMA complexity that can be used to prove strong $\widehat{\mathrm{MA}}^{\rightarrow}$ lower bounds for multiple problems?* It would then serve as a *canonical* hard function for OMA in this sense.
- *Can we exhibit an explicit function f with $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ (strongly) superlinear in $R^{\rightarrow}(f)$?* This would mean that the trivial OMA protocol is *significantly* better than any non-trivial protocol in terms of total communication.
- *If the answer to the above question is "yes", how large can this gap be—can it be exponential?* While it would be interesting to know that the minimum help needed from Merlin to reduce communication between Alice and Bob can be exponential in the classical one-way complexity, it would also show a contrast from the OMA model's version that allows Merlin and Alice to share randomness, where one can prove that a linear help message always suffices (see Observation 9.2.5 for details).
- *Can we show an explicit function f with $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ (strongly) superlinear in $\sqrt{N}$ for input size N?* This would mean that while there is a strong $\sqrt{N}$-barrier for $\mathrm{MA}^{\rightarrow}$ complexity [Gav21], the situation is not at all similar for $\widehat{\mathrm{MA}}^{\rightarrow}$.

In this work, we answer all these questions in the affirmative. (detailed discussions in Section 9.1.2).

Annotated Streaming. Next, we consider the analogous stream verification or *annotated streaming* model [CCM09, CCMT14], where a space-restricted Verifier and an all-powerful Prover with unlimited space simultaneously receive a huge data stream. Following the input stream, the Prover (with knowledge of the entire stream) and Verifier (who could only store a summary) invoke a *scheme* where the Prover tries to convince the Verifier about the answer to an underlying problem, similar to the online MA model. The total cost of a scheme is defined as the sum of the number of bits communicated by the Prover (hcost) and the number of bits of space used by the Verifier (vcost). The *non-trivial* annotated-streaming complexity of a function f is analogously defined as the minimum total cost over all non-trivial schemes computing f, where a trivial scheme is one that uses as much space (up to polylogarithmic factors) as is needed in classical streaming (without Prover).

Since all known lower bounds in annotated streaming are proven via reduction from problems in OMA communication, our knowledge of lower bounds in the two models are similar. We use our OMA results to prove strong lower bounds on the non-trivial complexity in this model as well. We show that fundamental data streaming problems such as *counting distinct items* have high non-trivial annotated streaming complexity when frequencies can be huge. Further, we show that graph problems such as connectivity and more generally k-connectivity have high non-trivial complexity under certain graph streams that we call *support graph turnstile* (SGT) streams. It might be intuitive that these problems might be hard in this model, even with a Prover. Perhaps surprisingly, we show that in the classical (sans Prover) model, we can solve these problems under SGT streams—featuring as large as exponential edge weights—almost as efficiently as under standard (unweighted) graph streams. We do this by building strong ℓ_0-samplers that can handle large frequencies and might be of independent interest. These results set the stage for our lower bounds on non-trivial complexity: they provide the benchmark space for any non-trivial scheme solving these problems.

Our final set of results give efficient annotated streaming schemes for k-connectivity on (standard) dynamic graph streams. We exploit graph theoretic properties on k-connected graphs to come up with short certificates for proving or disproving k-connectedness. This might be of independent interest in the graph-theoretic literature. Furthermore, these results establish a conceptual separation between classical streaming and annotated streaming: in the former, graph connectivity problems have roughly the same complexity in dynamic and SGT streams, whereas in the latter, they are much harder on SGT streams

than on dynamic. We discuss our annotated streaming results in detail in Section 9.1.2.

Basic Terminology. For the remainder of Section 1, it helps to define some basic terminology for ease of presentation. Later, in Section 9.2, we describe all notation and terminology in detail. An OMA protocol with hcost $O(h)$ (resp. $\widetilde{O}(h)$) and vcost $O(v)$ (resp. $\widetilde{O}(v)$) is called an (h, v)-OMA-protocol (resp. $[h, v]$-OMA-protocol). Analogously, an annotated streaming protocol is called an (h, v)-scheme or an $[h, v]$-scheme. For computing a function f, a *trivial OMA protocol* is one that has vcost $\Omega(R^{\rightarrow}(f))$, and a *trivial scheme* is one that uses verification space $\tilde{\Omega}(S(f))$, where $S(f)$ is the classical streaming complexity of f (the asymptotically optimal space for computing f in classical streaming). The "Index" communication problem and its variants come up frequently in our discussions. In the standard version IDX_N, Alice has a string $x \in \{0,1\}^N$ and Bob has an index $j \in [N]$, where his goal is to output $x[j]$, the jth bit of x.

9.1.2 Our Results and Contributions

First, we define the Equals-Index (henceforth, EQ-IDX) problem, which is the basis of our lower bounds.

Definition 9.1.1 (The EQ-IDX Problem)**.** *For arbitrary natural numbers p and q, in the $\text{EQ-IDX}_{p,q}$ communication problem, Alice holds strings $x_1, \ldots, x_p$ where each $x_i \in \{0,1\}^q$. Bob holds a string $y \in \{0,1\}^q$ and an index $j \in [p]$. The goal is for Bob to output whether $x_j = y$.*

This problem can be interpreted as (a boolean version of) the Index problem on large domains: Alice has a p-length string over the domain $\{0, \ldots, 2^q - 1\}$ (instead of just $\{0,1\}$). Bob needs to verify whether the jth index of Alice's string equals his value y.

Our main result on the non-trivial-OMA complexity of EQ-IDX is as follows.

Theorem 17. *For any p, q with $p = \Omega(\log q)$, we have $\widehat{\text{MA}}^{\rightarrow}(\text{EQ-IDX}_{p,q}) = \omega(q)$*

Observe that q can be as large as $\exp(\Omega(p))$. We also show that $R^{\rightarrow}(\text{EQ-IDX}_{p,q})$ is only $\Theta(p + \log q)$ (Lemma 9.4.1). This immediately implies that the gap between $\widehat{\text{MA}}^{\rightarrow}(f)$ and $R^{\rightarrow}(f)$ (or $\text{MA}^{\rightarrow}(f)$) can be exponential.

Corollary 9.1.2. *There is an explicit function f with*

$$\widehat{\text{MA}}^{\rightarrow}(f) = \exp(\Omega(\text{MA}^{\rightarrow}(f))) \text{ and } \widehat{\text{MA}}^{\rightarrow}(f) = \exp(\Omega(R^{\rightarrow}(f))).$$

Conceptually, this means that the trivial protocol where Merlin sends nothing and Alice and Bob solve the problem on their own, is exponentially better (in terms of total communication) than any other protocol[2].

Recall that previously we did not know of any function f on input size N with $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \tilde{\omega}(\sqrt{N})$. While exhibiting a function f with (standard) OMA complexity $\mathrm{MA}^{\rightarrow}(f) = \omega(\sqrt{N})$ remains a longstanding open problem, our results show that $\widehat{\mathrm{MA}}^{\rightarrow}$ complexity does not have such a barrier. In fact, Theorem 17 implies that it can be as large as $N/\log N$.

Corollary 9.1.3. *For any $C \in (\sqrt{N}, N/\log N)$, there is an explicit function f on input size N with*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(C).$$

Next, we turn to the related *Sparse Index* problem (henceforth SP-IDX). Chakrabarti et al. [CCGT14] defined the $\textsc{sp-idx}_{M,N}$ problem as the version of Index where Alice's string is promised to have hamming weight (i.e., number of 1's) at most M. They identified $\textsc{sp-idx}_{M,N}$ for $M = \log N$ as the first problem whose non-trivial OMA complexity is nearly as large as its one-way complexity.[3] Via reduction from EQ-IDX, we improve upon [CCGT14]'s lower bound for $\textsc{sp-idx}_{M,N}$ (for any M). In particular, our improved lower bound implies that SP-IDX with sparsity $O(\log\log N)$ has non-trivial-OMA complexity *exponential* in its classical one-way complexity.

Theorem 18. *For $M = \log\log N$, we have $\widehat{\mathrm{MA}}^{\rightarrow}(\textsc{sp-idx}_{M,N}) = \omega(\log N)$, whereas $R^{\rightarrow}(\textsc{sp-idx}_{M,N}) = \mathrm{MA}^{\rightarrow}(\textsc{sp-idx}_{M,N}) = \Theta(\log\log N)$.*

We remark that en route to establishing the above result, we also improve upon [CCGT14]'s upper bound on $R^{\rightarrow}(\textsc{sp-idx}_{M,N})$ and settle its complexity in the classical one-way model (see Lemma 9.4.3).

The only other functions (to the best of our knowledge) that prior work has shown to have OMA complexity (nearly) linear in its one-way complexity are the $\textsc{xor-conn}_n$ and $\textsc{xor-bip}_n$ problems [Tha16b]: in these problems, Alice and Bob have one graph each on the same vertex set $[n]$ (hence, the input size $N = \Theta(n^2)$), and they need to check connectivity and bipartiteness (respectively) of the graph obtained by XOR-ing their graphs, i.e., the

[2]Here, we are discounting clearly-suboptimal protocols where Alice sends $\omega(R^{\rightarrow}(f))$ bits or where Merlin sends a non-empty message despite Alice sending $\Omega(R^{\rightarrow}(f))$ bits. Hence, "any other protocol" essentially means any non-trivial protocol.

[3]It follows implicitly from their result establishing $\mathrm{MA}^{\rightarrow}(\textsc{sp-idx}_{\log N,N}) = \tilde{\Omega}(R^{\rightarrow}(\textsc{sp-idx}_{\log N,N}))$

graph induced by the symmetric difference of their edge sets. Thaler [Tha16b] showed that each of these functions f have $\mathrm{MA}^{\rightarrow}(f) = \Omega(n) = \tilde{\Omega}(R^{\rightarrow}(f))$, which implies the same about its $\widehat{\mathrm{MA}}^{\rightarrow}$ complexity. We reduce from EQ-IDX to reproduce this result, thus making a convincing case for EQ-IDX being a canonical problem for establishing high $\widehat{\mathrm{MA}}^{\rightarrow}$ complexity.

Corollary 9.1.4. *([Tha16b], paraphrased) For $f \equiv$ XOR-CONN$_n$ or $f \equiv$ XOR-BIP$_n$, we have*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(R^{\rightarrow}(f)) = \omega(n).$$

We now turn to the annotated streaming model. Our OMA results can be used to prove lower bounds on the total cost of any non-trivial scheme for certain problems.

First, consider the fundamental distinct items problem (henceforth, DIST-ITEM$_{N,F}$) on turnstile streams, where frequencies of elements from universe $[N]$ get incremented and decremented, and we are promised that the absolute value of the max-frequency is bounded above by F. At the end of the stream, we need to output the number of elements with non-zero frequency. We show a separation between classical streaming and (non-trivial) annotated streaming complexities for this problem, given as follows.

Theorem 19. *There is a setting of F such that* DIST-ITEM$_{N,F}$ *can be solved in $\widetilde{O}(N)$ space in classical streaming, but any non-trivial annotated streaming scheme for the problem must have total cost $\Omega(N^{\mathrm{polylog}\,(N)})$.*

Next, we show a similar separation for graph streaming problems. We define a support graph turnstile (SGT) stream to be one where an n-node graph is induced by the support of the edge-frequency vector. The parameter α of the SGT stream is the maximum possible absolute frequency. For the (undirected) graph connectivity problem, where we need to check if all pairs of nodes are reachable from each other, and the k-vertex-connectivity (resp. k-edge-connectivity) problem, where we need to check whether removal of some $k-1$ vertices (resp. edges) disconnects the graph, we show an $\widetilde{O}(n)$ vs $\Omega(n^{\mathrm{polylog}\,(n)})$ separation between classical streaming and non-trivial annotated streaming complexities of these problems under SGT streams.

Theorem 20. *Under certain support graph turnstile streams on n-node graphs, connectivity can be solved in $\widetilde{O}(n)$ space by a classical streaming algorithm, whereas any non-trivial annotated streaming scheme for the problem must have total cost $\Omega(n^{\mathrm{polylog}\,(n)})$.*

Theorem 21. *Under certain support graph turnstile streams on n-node graphs, k-vertex-connectivity and k-edge-connectivity can be solved in $\widetilde{O}(kn)$ space by a classical streaming*

algorithm, whereas any non-trivial annotated streaming scheme for the problem must have total cost $\Omega(n^{\text{polylog}(n)})$.

We remark that while the classical streaming space bounds of $\widetilde{O}(n)$ for connectivity and $\widetilde{O}(kn)$ for k-connectivity are known for standard dynamic graph streams [AGM12a, AS23a] where edge multiplicities are 0 or 1 throughout the stream, it was not known whether the same can be achieved for the harder SGT streams. We establish these upper bounds by designing ℓ_0-samplers that can handle frequency exponential in n while incurring just polylogarithmic factors in space. These might be of independent interest (see Lemma 9.5.4).

Finally, we design efficient schemes for k-vertex-connectivity and k-edge-connectivity under standard dynamic graphs streams. These schemes have total cost significantly smaller than the lower bound proven for schemes processing SGT streams.

Theorem 22. *Under dynamic graph streams on n-node graphs, there exists a $[k\cdot(h+kn), v]$-scheme for k-vertex-connectivity for any h, v such that $h \cdot v = n^2$. In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(k^2 n)$.*

Theorem 23. *Under dynamic graph streams on n-node graphs, there exists a $[k^2 n + h, v]$-scheme for k-edge-connectivity for any h, v such that $h \cdot v = n^2$. In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(k^2 n)$.*

Theorem 24. *Under dynamic graph streams on n-node graphs, there exists an $[n, n]$-scheme for k-edge-connectivity (for any k). In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(n)$.*

Contrast this with our results for annotated streaming under SGT streams, where the total cost for these problems can be as large as $\Omega(n^{\text{polylog}(n)})$. Thus, we can see a conceptual separation between classical streaming and annotated streaming: the former can tolerate SGT streams incurring negligible factors in complexity over dynamic graph streams, whereas the latter incurs significantly large factors for SGT streams over dynamic.

9.1.3 Related Work

In their seminal paper, Babai, Frankl, and Simon [BFS86] defined communication classes similar to classes in computational complexity; this included the Merlin-Arthur (MA) communication class, which essentially defined the Merlin-Arthur communication model. Klauck [Kla03] proved that disjointness and inner product have MA complexity $\Omega(\sqrt{N})$,

which was surprisingly proven to be tight (up to logarithmic factors) by Aaronson and Wigderson [AW08] who gave a protocol with total cost $O(\sqrt{N \log N})$. Chen [Che20] recently improved this bound to $O(\sqrt{N \log\log\log N})$. Chakrabarti et al. [CCMT14] were the first to consider the online version of the MA model. They used it to show lower bounds for annotated streaming schemes, as has been traditionally done by subsequent works. They proved that for any function f, an $[h, v]$-OMA-protocol that computes it must have $h \cdot v \geqslant R^{\rightarrow}(f)$. Further, for many problems including frequency moments and subset checks, they gave annotated streaming schemes (which imply OMA protocols with the same bounds), achieving this smooth tradeoff. [CCGT14] were the first to exhibit a problem, namely sparse index, where such a tradeoff is not possible. In fact, they showed that for sparse index with sparsity logarithmic in the input size, the OMA complexity is as large as the one-way communication complexity. Later, Thaler [Tha16b] exhibited two more problems with this property: XOR-connectivity and XOR-bipartiteness. The motivation was to show the existence of graph problems that provably need semi-streaming schemes (an $[n, n]$-scheme for n-node graphs) to solve and that they are equally hard in the annotated streaming model as in classical streaming.

An "augmented" version of the Equals-Index was studied by Jayram and Woodruff [JW13] in the classical communication model. They called it the "Augmented Index problem on large domains": here Bob also knows all the entries of Alice's vector before his input index.

For the problem of computing distinct items, there is an $(n^{2/3}(\log n)^{4/3}, n^{2/3}(\log n)^{4/3})$-scheme [CCMT14], which was later simplified and improved to an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme [Gho20]. Note that both of these works assume that the stream length m is $O(N)$, where N is the universe size. Our results are for streams that are exponentially longer.

With the growing interest in graph streaming algorithms over the last couple of decades, much of the recent literature on stream verification has focused on graph problems [CMT13, CCGT14, ADRV16, Tha16b, CG19, CGT20]. Most of them design stream verification protocols for insert-only or insert-delete graph streams. For the graph connectivity problem on n-node graphs, [CCMT14] gave an $[h, v]$-scheme for any h, v with $h \geqslant n$ and $h \cdot v = n^2$. For sparse graphs with m edges, [CCGT14] designed an $[n + m/\sqrt{v}, v]$-scheme. As mentioned above, [Tha16b] studied the problem in the XOR-edge-update model and proved that any $[h, v]$-scheme must have $(h + n) \cdot v \geqslant n^2$. A number of works [CMT13, CG19, CGT20] studied verification schemes for shortest-path and s, t-connectivity related problems. No prior work on stream verification, however, studied the k-connectivity problem.

In the classical streaming model, [AGM12a] gave the first algorithm for k-edge-connectivity in dynamic streams using $\widetilde{O}(kn)$ space. [GMT15] gave the first algorithm for k-vertex-

connectivity in dynamic graph streams, which was improved by a factor of k and made nearly optimal by [AS23a] using $\widetilde{O}(kn)$ space. [SW15] proved a lower bound of $\Omega(kn)$ bits for both problems, even for insertion-only streams (see also [AS23a] for extending the lower bound for k-vertex-connectivity to multiple passes).

Other variants of stream verification include a *prescient* setting where Prover knows the entire stream upfront, i.e., before Verifies sees it, and can send help messages accordingly [CCMT14, CCGT14]. Versions where Prover sends very large proofs have also been considered [KP13]. Natural generalizations to allow multiple rounds of interaction between the Prover and Verifier have been investigated. These include *Arthur-Merlin streaming protocols* of Gur and Raz [GR13] and the *streaming interactive proofs* (SIP) of Cormode et al. [CTY11]. The latter setting was further studied by multiple works [ADRV16, CCM+15, KP14]. Very recently, the notion of *streaming zero-knowledge proofs* has been explored [CDGH24]. For a more detailed survey of this area, see [Tha16a].

9.1.4 Technical Overview

Here, we give a high-level overview of our techniques and proof strategies. Some parts are simplified and corner cases are often skipped to convey the main essence of the techniques.

Communication Lower Bounds

The eq-idx lower bound. The basis of all our lower bounds is an OMA lower bound on the EQ-IDX problem (Definition 9.1.1). Recall the classical Index problem IDX_N where Alice has a string x of length N and Bob has an index $j \in [N]$ such that he needs to output $x[j]$. For any p, q with $p \cdot q = N$, Chakrabarti et al. [CCMT14] gave a (q, p)-OMA-protocol as follows. The string x can be partitioned into p chunks of length q each. Merlin sends Bob the purported chunk where j lies, thereby sending q bits. Again, Alice sends Bob a $\Theta(1)$-size equality sketch for each chunk, thereby sending $\Theta(p)$ bits. Using the relevant sketch, Bob figures out (w.h.p.) whether the chunk sent by Merlin is accurate and finds the solution if it is.

Note that from Alice's perspective, the problem actually boils down to the following subproblem: Alice has a string x partitioned into chunks $x_1, \ldots, x_p$ and Bob has a string y; she needs to help him verify whether y is identical to her k^{th} chunk, where she doesn't know k. Our main observation is that in the above protocol, Alice solves this subproblem using as many bits as she would have *without* Merlin: $\Theta(p)$ bits (we can show that this is tight for the problem under classical one-way communication). So now that Alice and Bob have Merlin, why not take his help and improve this communication to $o(p)$? If they

could do this with at most $O(q)$ bits of help, then they would obtain an improved $(q, o(p))$-OMA protocol for IDX_{pq}. But the known lower bound for IDX_{pq} [CCMT14] says that the product of hcost and vcost must be $\Omega(pq)$. Hence, Merlin cannot bring down the vcost to $o(p)$ without sending $\omega(q)$ bits. Since $\Theta(p)$, as noted, is the communication needed for the subproblem in the classical one-way model, we get that the non-trivial OMA complexity of the subproblem is $\omega(q)$. But q can be much larger than p—even exponential in p. Hence, we identify a problem whose non-trivial OMA complexity can be significantly larger than their one-way complexity. We essentially abstract out this subproblem as the $\text{EQ-IDX}_{p,q}$ problem and formalize the reduction.

The sparse index lower bound. Although this lower bound follows by a reduction from EQ-IDX, we discuss it separately to point out the challenges in proving its non-trivial-OMA complexity. Recall that the Sparse Index problem $\text{SP-IDX}_{M,N}$ is the version of IDX_N where Alice's string is promised to have at most M 1's. We prove that $\widehat{\text{MA}}^{\rightarrow}(\text{SP-IDX}_{\log\log N,N})$ is $\Omega(\log N)$, while $R^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) = \Theta(\log\log N)$, thus establishing yet another exponential gap.

Let us discuss the classical one-way complexity first. Chakrabarti et al. [CCGT14] showed that for any M, $R^{\rightarrow}(\text{SP-IDX}_{M,N}) = O(M\log M + \log N)$, but it was not known to be tight. We improve it to a tight bound of $\Theta(M + \log\log N)$, and then setting $M = \log\log N$ gives the desired bound. This improvement is the most challenging part in this result. The protocol of [CCGT14] works as follows. Alice picks a random function $h : [N] \to [M^3]$ from a pairwise independent hash family and sends Bob h along with the set of (at most) M values $h(i)$ for the M indices i where her string has a 1. Bob checks if $h(j)$ is in this set, where j is his input index, and if so, announces the answer to be 1, and 0 otherwise. The protocol can err only if Alice has a 0 at index j but $h(j)$ collides with one of the M $h(i)$'s, which happens with low probability by pairwise independence and union bound over the M elements. The set of M hash values can be expressed using $O(M\log M)$ bits and the function h takes $O(\log N)$ bits, giving their bound of $O(M\log M + \log N)$.

It is not hard to see that the $\log M$ factor can be removed with a tighter analysis: we reduce the range of the hash family from $[M^3]$ to $[3M]$. By pairwise independence, $h(j)$ collides with a single $h(i)$ with probability $1/3M$. Since we need to take union bound over at most m such $h(i)$'s and are happy with error probability at most $1/3$, this family suffices. Further, expressing the set of M $h(i)$'s then takes only $O(M)$ bits since Alice can simply send the set's characteristic vector of size $3M$. It is, however, not clear that we can reduce the additive $O(\log N)$ any further: it appears because the smallest known size of a pairwise independent hash family with domain $[N]$ is $\Omega(N)$. We, however, observe that $\text{SP-IDX}_{M,N}$ actually has input size $O(\binom{N}{M})$. Hence, if we run the above protocol but

draw the hash function using public randomness, thus obtaining an $O(M)$-cost public coin protocol, and then convert it to a private coin protocol using Newman's theorem, we incur an additive factor of $O(\log\log\binom{N}{M}) = O(\log M + \log\log N)$. Therefore, we prove that $R^{\rightarrow}(\text{SP-IDX}_{M,N}) = O(M + \log\log N)$. The lower bounds of $\Omega(M)$ and $\Omega(\log\log N)$ easily follow from well-known lower bounds of the (standard) index and equality functions, establishing the tight bound of $\Theta(M + \log\log N)$.

The above bound says that for any $M \geqslant \log\log N$, we have $R^{\rightarrow}(\text{SP-IDX}_{M,N}) = \Theta(M)$, meaning that any non-trivial protocol for the problem must have $\mathsf{vcost} = o(M)$. To prove that the non-trivial complexity $\widehat{\text{MA}}^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) = \Omega(\log N)$, we reduce $\text{SP-IDX}_{M,N}$ from $\text{EQ-IDX}_{p,q}$ with $p = M$ and $q = \log(N/M)$. Then, it follows that to obtain $\mathsf{vcost} = o(M)$, we need hcost of $\Omega(q) = \Omega(\log(N/M))$, which is $\Omega(\log N)$ for $M = \log\log N$. The reduction is very similar to the one in [CCGT14]. They reduced from IDX, while we reduce from EQ-IDX: we essentially modify their construction to fit the structure of the EQ-IDX problem.

Connectivity and bipartiteness on XOR-graphs. Via reduction from EQ-IDX, we reproduce the lower bounds for the other two problems that we know have high non-trivial OMA complexity. These are the XOR-CONN_n and XOR-BIP_n problems [Tha16b] that respectively ask whether the XOR of the graphs that Alice and Bob hold on the vertex set $[n]$, is connected and bipartite. Our reduction is very similar to those by Thaler [Tha16b]. Although he reduced from IDX_{n^2} and we reduce from $\text{EQ-IDX}_{n,n}$, our construction can be seen as adapting his construction to the structure of the EQ-IDX problem.

The reduction from $\text{EQ-IDX}_{n,n}$ to XOR-CONN_{2n+1} works as follows. Given the input strings $x_1, \ldots, x_n$ where each $x_i \in \{0,1\}^n$, Alice constructs a bipartite graph with n nodes on each partite set: for each $i \in [n]$, she sees x_i as the characteristic vector of the neighborhood of ℓ_i, the i^{th} vertex on the left. To ensure that her graph is connected, she joins all vertices to a dummy vertex, say v. Bob treats his input y as the characteristic vector of a set of edges incident to ℓ_j. If y is indeed identical to x_j, then XOR of these edges and the edge $\{(\ell_j, v)\}$ with the original graph must isolate ℓ_j. Hence, the XOR of Alice's and Bob's graphs is disconnected in this case. Otherwise, in the XOR graph, ℓ_j must be adjacent to some vertex, say w, on the right; again, w, in turn, is adjacent to v. Since all other vertices are adjacent to v, this ensures that the graph is connected in this case. Thus solving XOR-CONN gives the answer to EQ-IDX, and we obtain the desired result on its non-trivial OMA complexity.

For XOR-BIP, the construction is very similar. After constructing the same base bipartite graph using the x_i's, Alice joins only the vertices on the right to the dummy vertex v. Bob constructs the same edge set using y. The XOR of these edges and ℓ_j, v with Alice's

graph creates a triangle if $y \neq x_j$: as before, ℓ_j must be adjacent to some w on the right, and both of them are adjacent to v. Hence, in this case, the graph is not bipartite. If y is indeed identical to x_j, then adding v and ℓ_j respectively to the left and the right partite sets of the original graph gives a bipartition of the XOR graph. Thus, XOR-BIP can be used to solve EQ-IDX, giving us the desired result on its non-trivial OMA complexity.

Annotated Streaming Lower Bounds

We reduce from EQ-IDX to prove lower bounds on the *non-trivial annotated streaming complexity* of several problems. It is defined as the minimum tcost over all annotated streaming schemes that solve the problem with space (i.e., vcost) sublinear in its classical streaming complexity (ignoring polylogarithmic factors). The standard line of argument that allows us to prove annotated streaming lower bounds via OMA communication complexity (similar to classical streaming lower bounds via classical communication complexity) is as follows. Given an annotated streaming problem, we assign part of the problem's input to Alice and the rest to Bob under the OMA setting. Given a scheme for the problem, Alice simulates Verifier and runs his algorithm on her part of the input. She then sends the memory state to Bob, who continues Verifier's algorithm by extending the stream with his part of the input. Merlin then simulates Prover in the scheme, and sends the proof to Bob, who uses the final memory state of Verifier's algorithm and the proof to obtain the answer. Thus, given an (h, v)-scheme for the problem, we can obtain an (h, v)-OMA protocol for its communication version. Hence, an OMA lower bound translates to an annotated streaming lower bound. Below, we describe the reductions from EQ-IDX to certain annotated streaming problems by stating how Alice and Bob construct their part of the input and how solving the respective problems on the combined input solves EQ-IDX. The annotated streaming lower bound then follows by the argument mentioned above.

Distinct items. First, we consider the fundamental *distinct items* problem DIST-ITEM$_{N,F}$, where we need to count the number of items with non-zero frequency as elements from the universe $[N]$ get inserted and deleted in a stream, and the absolute value of each frequency is bounded above by F. Let us first see how we can solve it in classical streaming. For each element, we keep a *decision counter*, which, at the end of the stream, reports whether or not the element has non-zero frequency. In this work, we build such counters using $\widetilde{O}(\log \log F)$ space each. Thus, the problem can be solved using $\widetilde{O}(N \log \log F)$ space in classical streaming. Therefore, even if we set $F = \exp(N^{\mathrm{polylog}\,(N)})$, we get that the classical streaming complexity is $\widetilde{O}(N)$. A lower bound of $\Omega(N)$ is easy to show. Hence, any non-trivial annotated streaming scheme for the problem must have $\mathsf{vcost} = o(N)$. Then we reduce DIST-ITEM$_{N,F}$ from EQ-IDX$_{p,q}$ for $p = N$ and $q = \log F$, implying that any non-trivial scheme must have $\mathsf{hcost} = \Omega(q) = \Omega(\log F)$. For $F = \exp(N^{\mathrm{polylog}\,(N)})$, this gives a

bound of $\Omega(N^{\text{polylog}\,(N)})$ on the hcost, and hence the tcost. Thus, we see that for a certain setting of F, the non-trivial annotated streaming complexity of DIST-ITEM$_{N,F}$ can be as large as $\Omega(N^{\text{polylog}\,(N)})$, while the classical streaming complexity is only $\widetilde{O}(N)$.

The reduction is straightforward: given the inputs $x_1, \ldots, x_N \in \{0,1\}^{\log F}$ in the EQ-IDX$_{N,\log F}$ problem, Alice treats the x_is as binary representations of numbers in $[0, F-1]$ and sets her part of the input stream as $x_i + 1$ insertions of the element i, for each $i \in [N]$. Bob sets his part of the stream as $y + 1$ deletions of the element j. It follows that this stream has N distinct items iff $x_j \neq y$. Hence, we can solve EQ-IDX$_{N,\log F}$ using a scheme for DIST-ITEM$_{N,F}$.

Connectivity-related problems on SGT streams. To establish separations between standard and non-trivial annotated streaming complexities for graph problems, we introduce the support graph turnstile (SGT) model where the input n-node graph is defined by the *support* of the $\binom{n}{2}$-length frequency vector whose entries, representing edge frequencies of the graph, are updated with insertions and deletions of edges in the stream. We reduce global connectivity and k-connectivity (both vertex and edge versions) under such streams from EQ-IDX to establish their high non-trivial annotated streaming complexity. To set the stage, we first prove that in the classical streaming model, the known upper bounds for these problems under the standard insert-only or dynamic streams can be matched under SGT streams at the cost of just polylogarithmic factors. This means global connectivity under SGT streams can be solved using $\widetilde{O}(n)$ space and both k-vertex-connectivity and k-edge-connectivity can be solved in $\widetilde{O}(kn)$ space. The crucial tool we use here are the strong ℓ_0-samplers that we build in this paper. We give an overview of this in the next subsection. Below, we assume the classical streaming complexity of these problems and discuss our reductions for establishing the non-trivial complexity.

For connectivity, we reduce from EQ-IDX$_{n,n\log F}$. Given her inputs $x_1, \ldots, x_n \in \{0,1\}^{n\log F}$, Alice constructs a bipartite graph with n nodes on each partite set. She treats each x_i as n blocks of size $\log F$ each, representing the edge frequencies (each value lying in $[F]$) of the n edges incident on the ith left vertex. Strong Bob then breaks y into n blocks $y_1, \ldots, y_n$ of $\log F$ bits each. For each $b \in [n]$, he deletes the edge between the jth left vertex and bth right vertex $y_b + 1$ times. Therefore, if $y = x_j$, then the jth left vertex gets isolated in the resultant support graph, and it is not connected. Otherwise, the support graph must have the jth left vertex connected to some node on the right, and the rest of the graph is a complete bipartite graph. Hence, the graph is connected. It follows that any non-trivial scheme for connectivity under SGT streams, i.e., a scheme with vcost $= o(n)$ necessiates hcost $= \Omega(n \log F)$. As before, setting $F = \exp(n^{\text{polylog}\,n})$, we get that the non-trivial streaming complexity of connectivity can be as large as $\Omega(n^{\text{polylog}\,n})$ while the

classical streaming complexity is $\widetilde{O}(n)$.

For k-connectivity, we reduce from EQ-IDX$_{kn,\log F}$. Given her inputs $x_1, \ldots, x_{kn} \in \{0,1\}^{\log F}$, Alice constructs a bipartite graph with k nodes on the left partite set and n nodes on the right partite set. She treats each $x_i \in \{0,1\}^{\log F}$ as the edge frequency (lying in $[F]$) of the ith edge (following some canonical ordering of the kn possible edges). Bob then deletes the jth edge y times (with $y \in \{0,1\}^{\log F}$ seen as a number in $[F]$). Thus, the resultant support graph is a complete bipartite if $x_j \neq y$. Otherwise, it's one edge short of a complete graph: namely the jth edge. It is easy to prove that the graph in the first case is k-vertex-connected as well as k-edge-connected, whereas the graph in the second case is neither. Hence, k-connectivity (edge or vertex version) on $\Theta(n)$ nodes under SGT streams can be used to solve EQ-IDX$_{kn,\log F}$. It follows that any non-trivial scheme, i.e., a scheme with $\mathsf{vcost} = o(kn)$ necessiates $\mathsf{hcost} = \Omega(\log F)$. Again setting $F = \exp(n^{\text{polylog}\, n})$, we get that the non-trivial streaming complexity of connectivity can be as large as $\Omega(n^{\text{polylog}\, n})$ while the classical streaming complexity is $\widetilde{O}(kn)$.

Classical Streaming Algorithms

The prerequisite of computing the non-trivial annotated streaming complexity of a function is knowing its classical streaming complexity. Hence, for each problem mentioned in Section 9.1.4, we need to establish its classical streaming complexity. This requires considerable work especially for the SGT problems since we introduce the SGT model in this work. Most of our algorithms for SGT streams follow by replacing the ℓ_0-samplers in existing dynamic-streaming algorithms by the new (strong) ℓ_0-samplers that we design in this work. However, an algorithm for SGT k-edge-connectivity does not immediately follow by just this replacement. So we design a new algorithm here, and this is one of our significant technical contributions. We discuss these techniques in detail below.

Strong ℓ_0-samplers. An *ℓ_0-sampler* runs on a stream's frequency vector and outputs an element uniformly at random from the support of the vector, i.e., the set of elements with non-zero frequency. We design new ℓ_0-samplers that can handle very large frequencies (hence called strong), say as large as $\exp(N^{\text{polylog}\, N})$ for universe $[N]$, and still take only $\text{polylog}\,(N)$ space. This is helpful because the SGT streams establishing our large separations have such frequencies. Standard $\text{polylog}\,(N)$-space ℓ_0-samplers usually assume that the frequencies are bounded by $\text{poly}(N)$.

The main tool we use to design our sampler is our *decision counter*. Given a stream of insertions and deletions, these counters detect whether or not the number of insertions is exactly equal to the number of deletions, using very little space. This even works when the difference between the number of insertions and deletions is very large, which is challenging

to do in small space deterministically. The idea is to maintain a standard counter modulo a random prime of value roughly $\log F$, where F is the maximum absolute frequency. It follows that with high probability, the counter value is non-zero iff the actual frequency is non-zero. Thus, the space needed by a decision counter is only $O(\log \log F)$.

We then use ideas from the sparse recovery and ℓ_0-sampling literature to build our strong ℓ_0-sampler. First we focus on the special case when the support size is exactly 1. The problem thus boils down to the following: given a stream of insertions and deletions over a universe of N elements and promised that at the end of the stream, there is exactly one element with non-zero frequency, we need to find that element.

We first point out why the standard data structure for this problem does not work for our purpose. It maintains one counter to track the sum of all frequencies f_i and another one to maintain the sum of $i \cdot f_i$, over all $i \in [N]$. Since the support size is promised to be 1, the first counter has f_j and the second one has $j \cdot f_j$ at the end of the stream, where j is the special element. Thus, we can easily figure out j from these two counters. Observe that the counters take $O(\log F + \log N)$ space in total. Hence, for our desired setting of $F = \exp(N^{\text{polylog}\,(N)})$—in fact, for any $F >> N^{\text{polylog}\,(N)}$—it uses much more than $\text{polylog}\,(N)$-space.

To tackle this, we simulate binary search using our decision counters. First suppose that we were allowed $\log N$ adaptive passes over the stream. Then, in the first pass, we can use our decision counter on the sub-stream given by only the first half of the universe and figure out which half the special element j lies in. Then, in each subsequent pass, we split the candidate set obtained so far into two halves and run the decision counter on the sub-stream induced on that half. Hence, we can figure out j after $\log N$ passes. To simulate this in a single pass non-adaptively, we can parallelly run $\log N$ decision counters, one on each of the $\log N$ vectors obtained as follows. We take the dot product of the stream's frequency vector with the following set of $\log N$ vectors. For $i \in [\log N]$, the ith vector breaks the universe $[N]$ into 2^i chunks, with the chunks being all 0's and all 1's alternately. It is not hard to see that we can simulate binary search using the outputs of the decision counters on these dot products. This strategy is called *non-adaptive* binary search in the literature. Note that the total space taken is $O(\log \log F \cdot \log N)$ since each decision counter takes $O(\log \log F)$ space. This is $\text{polylog}\,(N)$ space even for our desired setting $F = \exp(N^{\text{polylog}\,(N)})$.

Now consider the general version with no promise on the support size. We reduce it to the support 1 case. Maintain $\log N$ parallel guesses of the support size in powers of 2 and for each guess s, sample the elements with probability $1/s$. We can show that for the closest guess of s, we sample just a single element from the support with constant probability. We

can repeat $O(\log N)$ times for high probability. Then running the algorithm for support 1 as a blackbox on the sampled sub-stream solves the ℓ_0-sampling problem.

Finally, we need to correctly identify whether a sampled sub-stream indeed has support 1, before we apply the blackbox on it. We design a sketch for this by randomly partitioning the universe into 4 parts, summing those parts up and checking how many parts are non-zero. The hope is that, with high probability, if there are at least 2 elements in the support, then multiple parts will become non-zero, indicating a support size > 1. If the support is 1, then exactly one part will be non-zero. If the support is 0, then all parts will be 0. We show that this happens with constant probability, and repeat it $O(\log N)$ times to boost the probability of success. Putting everything together gives us the strong ℓ_0-sampler sketch in $\widetilde{O}(\log \log F)$ space.

Now using our new decision counters for the DIST-ITEM$_{N,F}$ problem gives an $\widetilde{O}(N)$-space classical streaming algorithm for our setting of F. Also, replacing the standard ℓ_0-samplers in the connectivity-sketch of [AGM12a] and k-vertex-connectivity algorithm of [AS23a] by our strong ℓ_0-samplers gives $\widetilde{O}(n)$-space and $\widetilde{O}(kn)$-space algorithms respectively for the problems under SGT streams.

Edge connectivity under SGT streams. Unlike k-vertex-connectivity, the state-of-the-art dynamic-streaming algorithm for k-edge-connectivity cannot be extended to SGT streams by simply replacing their ℓ_0-samplers with strong ℓ_0-samplers. This algorithm by [AGM12a] uses $\widetilde{O}(kn)$ space and essentially builds a certificate of that size, i.e., a spanning $\widetilde{O}(kn)$-sized subgraph with the same answer to k-edge-connectivity as the original graph. The certificate is essentially the union of k edge-disjoint spanning forests. After recovering one spanning forest T_1 of G, they delete the edges of T_1 from G and then recover a new spanning forest T_2 (so that it's disjoint from T_1). This is easy to do in dynamic streams because the frequencies of the edges of T_1 are exactly 1. So we can extend the stream with deletions of the edges of T_1 and obtain $G - T_1$. However, in SGT streams, the frequency of the edges of T_1 could be arbitrary, and the recovery of T_1 using ℓ_0-samplers do not reveal the exact frequencies of its edges. So we cannot easily delete T_1 from G in order to recover T_2. This dependency between the spanning forests makes extending this algorithm difficult. We design a new algorithm that gets rid of this dependency and is thus extendable to SGT streams.

Our randomized algorithm obtains a certificate for k-edge-connectivity of size $\widetilde{O}(kn)$, which is optimal up to polylog factors. We show that using our strong ℓ_0-samplers and the spanning forest recovery technique of [AGM12a], we can construct the certificate under SGT streams. Our algorithm for building the certificate is heavily inspired by [AS23a]'s algorithm for *k-vertex-connectivity* certificate. We independently sample every edge with

probability $1/k$ and find a spanning forest of the sampled subgraph. The certificate then is a union of the spanning forests in $\widetilde{O}(k)$ such independent iterations. The analysis then shows that edges whose endpoints are not very well connected in the original graph exist in the certificate, and pairs of vertices that are very well connected in the original graph are well connected in the certificate. This is enough to prove that the certificate preserves the answer to k-edge connectivity.

Schemes for k-connectivity under dynamic streams

We design schemes for vertex connectivity and edge connectivity under dynamic streams to show that they are much easier on such streams than on SGT streams in the annotated streaming model (unlike the classical streaming model where they have roughly the same complexity). We discuss the tools and techniques for these schemes below.

Vertex connectivity. We give a $[k^2n, n/k]$-scheme for k-vertex-connectivity under dynamic streams of n-node graphs. A key technical ingredient is our "layering lemma", which says that there exists a short proof (size $\widetilde{O}(kn)$) to show that any arbitrary fixed vertex has k vertex-disjoint paths to all other vertices. This proof can also be verified in $\widetilde{O}(n/k)$ space. Also, when proving this for r vertices, we can reuse space and thus get an $[r \cdot kn, n/k]$-scheme. If we prove it for k vertices, then using properties of k-vertex-connectivity, we can prove that the graph is k-vertex-connected. Hence, we get a $[k^2n, n/k]$-scheme. We also extend these ideas to k-edge-connectivity to obtain a $[k^2n, n/k^2]$-scheme.

To show the existence of the short proof given by the layering lemma, we use probabilistic methods. Recall that we want to show that a given vertex has k vertex-disjoint paths to all other vertices. The idea is to cleverly partition the vertex set into $\log n$ layers and show that vertices in the first layer have k vertex-disjoint paths to the given vertex. Then we inductively show that vertices in a layer have k vertex-disjoint path to the previous layer. Using the properties of k-vertex-connectivity, this is enough to show that the special vertex has k vertex-disjoint paths to all other vertices. The proof for showing connectivity between any two layers is of size $\widetilde{O}(kn)$, giving us the desired bound. The verification is more involved, but the Verifier essentially checks two things: (i) whether the edges sent by the prover belong to the input graph, and (ii) if what the prover sent are indeed vertex-disjoint paths. For this, we use standard subroutines in the annotated streaming literature like subset-check and duplicate-detection schemes.

Edge connectivity. We present an $[n, n]$-scheme for k-edge-connectivity under dynamic streams of n-node graphs. We achieve this by simulating the two-pass classical-streaming algorithm for global minimum cut, implicit in [RSW18], in a single annotated-streaming

pass. During the stream, Verifier computes a cut sparsifier of the graph. After the stream, he compresses the vertices into supernodes (by compressing all large cuts) such that only the small cuts, inlcuding the min-cut, remain. Prover then sends all the edges of this contracted graph: [RSW18] showed that there are only $O(n)$ such edges. Verifier can then compute the exact min-cut of this graph, and in turn, of the original graph, since the min-cut is preserved in the contracted graph. Consequently, Verifier can solve k-edge-connectivity for all values of k.

9.1.5 Paper Organization

We start by defining the models and notation in Section 9.2. Then in Section 9.3, we list all preliminary facts we use throughout the paper. In Section 9.4, we formally introduce our central problem called Equals-Index and use it to prove lower bounds for several problems in OMA communication complexity and annotated streaming. Section 9.5 is our toolkit section where we build our algorithm for the strong ℓ_0-samplers. We then provide an algorithm for k-edge-connectivity in the classical streaming model under both dynamic and SGT streams. The techniques in Section 9.5 can be used to get SGT streaming algorithms for connectivity and k-veretex-connectivity. In Section 9.6, we present schemes for k-vertex-connectivity and k-edge-connectivity under dynamic streams. The layering lemma, the primary tool for our k-vertex-connectivity protocol, is proven there. Finally, in Section 9.7, we give a scheme for k-vertex-connectivity under SGT streams.

9.2 Models, Notation, and Terminology

9.2.1 Models

We formally describe the various communication and streaming models that we focus on in this paper and define the notation and terminology in each model.

Communication Models

Classical Two-Party Communication. In the (randomized) two-party communication model introduced by Yao [Yao79], two players Alice and Bob possess inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively. Their goal is to compute $f(x, y)$, where $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$. To this end, they use a communication protocol Π where they toss some random coins and accordingly send messages to each other back and forth in rounds. The protocol Π terminates when one of the players announces an output. In this paper, we only focus on one-way communication, i.e., when Π has just a single round: Alice sends a single message

to Bob, from which he must declare the output. We now describe this variant of the model in more detail.

One-way randomized communication. In a *private coin* protocol, Alice's coin tosses are private, i.e., she draws a random string $\mathcal{R}_A$ that is unknown to Bob, and sends him a message msg as a function of $(x, \mathcal{R}_A)$. After receiving Alice's message, Bob draws a random string $\mathcal{R}_B$ and outputs a bit $\mathsf{out}(\Pi)$ as a function of $(y, \mathsf{msg}, \mathcal{R}_B)$. In a *public coin* protocol, $\mathcal{R}_A$ is also known to Bob (without any communication).

We say that a protocol Π for a function f has error δ if $\Pr_{\mathcal{R}_A,\mathcal{R}_B}[\mathsf{out}(\Pi) \neq f(x,y)] \leqslant \delta$. We say that a protocol Π *solves* a function f if it has error $\delta \leqslant 1/3$.

The communication cost $\mathrm{CC}(\Pi)$ of protocol Π is the maximum length of Alice's message over all possible $(x, \mathcal{R}_A)$. The randomized one-way communication complexity of a function is defined as

$$R^{\rightarrow}(f) := \min\{\mathrm{CC}(\Pi) : \Pi \text{ is a private coin protocol that solves } f\}$$

When public randomness is allowed, we denote this complexity as

$$R^{\rightarrow(\mathrm{pub})}(f) := \min\{\mathrm{CC}(\Pi) : \Pi \text{ is a public coin protocol that solves } f\}$$

Merlin-Arthur Communication. In the Merlin-Arthur (MA) communication model [BFS86], we have the usual two players Alice and Bob (collectively called "Arthur") with their inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively. They want to compute $f(x,y)$, where $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$. In addition, there is an all-powerful player Merlin who knows the inputs x and y. Merlin is untrusted; he sends Alice and Bob a "proof" of the solution, following which these two players interact between themselves to verify the proof. In this paper, we focus on the *online Merlin-Arthur* (OMA) model which is an MA analog of the one-way randomized communication model defined above. We describe this model in more detail.

Online Merlin-Arthur communication. An OMA protocol Π works as follows. Merlin sends Bob a help message $\mathcal{H}$. Then Alice generates a random string $\mathcal{R}_A$, based on which she sends Bob a message msg. Bob then outputs $\mathsf{out}(\Pi) \in \{0, 1, \perp\}$ as a function of $(y, \mathcal{H}, \mathsf{msg}, \mathcal{R}_B)$, where $\mathcal{R}_B$ is his private random string. Indeed, there is no shared randomness between Merlin and Arthur (Alice and Bob). An OMA protocol Π is said to have completeness error δ_c and soundness error δ_s if the following conditions are satisfied.

- (Completeness) If $f(x,y) = 1$, then there exists a function $\mathcal{H}$ such that $\Pr_{\mathcal{R}_A,\mathcal{R}_B}[\mathsf{out}(\Pi) \neq 1] \leqslant \delta_c$

- (Soundness) If $f(x,y)=0$, then $\forall \mathcal{H}' : \Pr_{\mathcal{R}_A,\mathcal{R}_B}[\mathsf{out}(\Pi)=1] \leqslant \delta_s$

We say that a protocol Π *solves* a function f if Π has $\delta_s, \delta_c \leqslant 1/3$ for computing f.

For an OMA protocol Π, the *help cost* $\mathsf{hcost}(\Pi)$ is the maximum length of $\mathcal{H}$ over all possible (x,y), and the *verification cost* $\mathsf{vcost}(\Pi)$ is the maximum number of bits sent by Alice over all possible $(x,\mathcal{R})$. The *total cost* $\mathsf{tcost}(\Pi)$ is then defined as $\mathsf{hcost}(\Pi)+\mathsf{vcost}(\Pi)$. The OMA-complexity of f is defined as

$$\mathrm{MA}^{\rightarrow}(f) := \min\{\mathsf{tcost}(\Pi) : \Pi \text{ solves } f\}.$$

Non-trivial OMA complexity and its basic analysis. We define the *non-trivial-OMA complexity* of a function in this paper. To this end, we first formally define trivial and non-trivial OMA protocols.

Definition 9.2.1 ((Non-)Trivial OMA protocol)**.** *We say that an OMA protocol Π solving a function f is* trivial *if $\mathsf{vcost}(\Pi) = \Omega(R^{\rightarrow}(f))$ and is* non-trivial *if $\mathsf{vcost}(\Pi) = o(R^{\rightarrow}(f))$.*

Definition 9.2.2 (Non-trivial OMA complexity)**.** *Let f be a function with $R^{\rightarrow}(f) = \omega(1)$. The non-trivial OMA complexity of f is given by*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) := \min\{\mathsf{tcost}(\Pi) : \Pi \textit{ is a non-trivial protocol that solves } f\}.$$

If f is a function with $R^{\rightarrow}(f) = \Theta(1)$, then $\widehat{\mathrm{MA}}^{\rightarrow}(f) := \infty$.

We remark that if $R^{\rightarrow}(f) = \Theta(1)$, then attaining $\mathsf{vcost} = o(R^{\rightarrow}(f))$ would mean having $\mathsf{vcost} = 0$, which can be shown to be impossible for any "interesting" function (whose value depends on both inputs). Hence, it makes sense to define $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ to be ∞ in this case. Next, we note that $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ is finite for any function f with $R^{\rightarrow}(f) = \omega(\log n)$.

Observation 9.2.3. If $R^{\rightarrow}(f) = \omega(\log n)$, then $\widehat{\mathrm{MA}}^{\rightarrow}(f) \leqslant O(N)$, where N is the input size.

Proof. Merlin purportedly sends Alice's entire input to Bob (costing $O(N)$ bits) while Alice sends an equality-sketch of size $O(\log n)$ (Fact 9.3.2) for her input. Provided the check goes through, Bob can figure out the solution from both inputs. Since $R^{\rightarrow}(f) = \omega(\log n)$ and this protocol has $\mathsf{vcost} = O(\log n)$, it is a non-trivial protocol with $\mathsf{tcost} \leqslant O(N)$. ∎

We leave open whether $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ can be unbounded if $R^{\rightarrow}(f)$ is $\omega(1)$ but $O(\log N)$.

We also make the following observations.

Observation 9.2.4. The following hold for any function f.

(i) If $\mathrm{MA}^{\rightarrow}(f) = o(R^{\rightarrow}(f))$, then $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \mathrm{MA}^{\rightarrow}(f)$.

(ii) If $\mathrm{MA}^{\rightarrow}(f) = \Omega(R^{\rightarrow}(f))$, then $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \Theta(H)$, where

$$H := \min\{\mathsf{hcost}(\Pi) : \text{protocol } \Pi \text{ solves } f \text{ with } \mathsf{vcost}(\Pi) = o(R^{\rightarrow}(f))\}$$

Proof. In case (i), $\min\{\mathsf{tcost}(\Pi) : \text{protocol } \Pi \text{ solves } f\} = o(R^{\rightarrow}(f))$ (by definition of $\mathrm{MA}^{\rightarrow}(f)$). Thus, the minimizing protocol Π^* must have $\mathsf{tcost}(\Pi^*) = o(R^{\rightarrow}(f))$, and hence $\mathsf{vcost}(\Pi^*) = o(R^{\rightarrow}(f))$. Therefore, Π^* is a non-trivial protocol. Then, by definition of $\widehat{\mathrm{MA}}^{\rightarrow}(f)$, we have $\widehat{\mathrm{MA}}^{\rightarrow}(f) \leqslant \mathsf{tcost}(\Pi^*) = \mathrm{MA}^{\rightarrow}(f)$. Again, $\mathrm{MA}^{\rightarrow}(f) \leqslant \widehat{\mathrm{MA}}^{\rightarrow}(f)$ for all functions. Hence, we conclude $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \mathrm{MA}^{\rightarrow}(f)$.

In case (ii), by definition of $\mathrm{MA}^{\rightarrow}(f)$, any protocol Π that solves f must have $\mathsf{tcost}(\Pi) = \Omega(R^{\rightarrow}(f)$. Since any non-trivial protocol Π for f has $\mathsf{vcost}(\Pi) = o(R^{\rightarrow}(f))$, then it must have $\mathsf{hcost}(\Pi) = \Omega(R^{\rightarrow}(f))$. Thus, $\mathsf{hcost}(\Pi) > \mathsf{vcost}(\Pi)$, implying that $\mathsf{tcost}(\Pi) = \mathsf{hcost}(\Pi) + \mathsf{vcost}(\Pi) = \Theta(\mathsf{hcost}(\Pi))$. The claim then follows by the definitions of $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ and non-trivial protocols.

Thus, Observation 9.2.4 (ii) shows that for functions f with $\mathrm{MA}^{\rightarrow}(f) = \Omega(R^{\rightarrow}(f))$ (ones that we will be mostly dealing with in this paper), the non-trivial OMA complexity can be interpreted as the minimum hcost needed to achieve $\mathsf{vcost} = o(R^{\rightarrow}(f))$.

Finally, we observe that if we allowed Merlin and Alice to share randomness, as in the AM model, then the non-trivial complexity would always be upper bounded by $O(R^{\rightarrow}(f))$. This means a help message from Merlin linear in $R^{\rightarrow}(f)$ always suffices in reducing communication between Alice and Bob. This is in sharp contrast with our results in the standard OMA model (disallowing sharing of random bits) that show that $\widehat{\mathrm{MA}}^{\rightarrow}(f)$ can be as large as $\exp(R^{\rightarrow}(f))$.

Observation 9.2.5. Suppose that Merlin and Alice could share randomness in the OMA model. Then $\widehat{\mathrm{MA}}^{\rightarrow}(f) = O(R^{\rightarrow}(f))$ for any function f.

Proof. Merlin and Alice use their shared randomness to draw a random string for the optimal classical one-way protocol Π that solves f. Then Merlin puportedly sends Bob the

same message msg of size $R^{\rightarrow}(f)$ that Alice would have sent in Π for that random string. Alice then sends Bob an equality-sketch of msg (using her private randomness) costing $O(\log(R^{\rightarrow}(f))) = o(R^{\rightarrow}(f))$ bits. Provided the check goes through, Bob can figure out the output by running Π using Merlin's message. Thus, we get a non-trivial protocol of $\mathsf{tcost} = O(R^{\rightarrow}(f))$. ∎

Streaming Models

First, we define the different types of data streams that we consider in this paper.

Data Streams. A data stream of length m is defined as a sequence $\sigma := \langle (a_i, \Delta_i) : i \in [m] \rangle$, where $a_i \in [N]$ and Δ_i is an integer. The data stream defines a *frequency vector* $\mathbf{f}(\sigma) := \langle \text{freq}(1), \ldots, \text{freq}(N) \rangle$, where

$$\text{freq}(j) := \sum_{i \in [m]:\ a_i = j} \Delta_i$$

is the frequency of element $j \in [N]$. In the streaming model, we need to compute some function of $\mathbf{f}(\sigma)$ for an input stream σ using space sublinear in the input size, i.e., $o(m)$ space.

Turnstile streams. If a stream is as defined above, with no restriction on Δ_i and $\mathbf{f}(\sigma)$, then we call it a turnstile stream. Conceptually, turnstile streams allow frequencies of elements to be incremented and decremented arbitrarily with each update, and the frequency vector can have negative entries.

Insert-only streams. A data stream is called insert-only if $\Delta_i > 0$ for all $i \in [m]$. A well-studied special case is the *unit update* model, where $\Delta_i = 1$ for all $i \in [m]$.

Insert-delete streams. We call a data stream an insert-delete stream if $\Delta_i \in \{-1, 1\}$ for all $i \in [m]$ and $\text{freq}(j) \in \{0, 1\}$ for all $j \in [N]$ at all points during the stream. Conceptually, elements are either inserted or deleted, frequency can be at most 1, and an element can be deleted only if its frequency is 1 during the update.

Graph Streams. A graph stream is a data stream whose frequency vector defines a graph. Here, $N = \binom{n}{2}$ where $[n]$ is the set of nodes of the graph. The frequency vector is indexed by pairs $(u, v) \in \binom{[n]}{2}$ (using a canonical bijection from $[N]$ to $\binom{[n]}{2}$), and typically, $\text{freq}(u, v)$ represents the weight of the edge $\{u, v\}$.

Dynamic graph streams. We call an insert-delete graph stream as a dynamic graph stream. Conceptually, the input graph is unweighted, and an edge can be deleted only if it is present in the graph during the update.

We now introduce support graph turnstile (SGT) streams. Conceptually, the graph is induced by the support of the frequency vector of the input stream, which may be a turnstile stream.

Definition 9.2.6 (Support graph turnstile streams)**.** *A turnstile stream σ is called a support graph turnstile stream if it is of the form $\langle (u,v)_i, \Delta_i) : i \in [m] \rangle$, where $(u,v) \in \binom{[n]}{2}$ and the graph that it defines is given by $G = ([n], \{(u,v) : \text{freq}(u,v) \neq 0\})$.*

Now we describe the streaming models that we study in this paper.

Classical Streaming. In the classical streaming model, a (randomized) data streaming algorithm $\mathcal{A}$ reads an input stream σ and needs to compute some function $f(\sigma)$ of the stream. To this end, it maintains a summary $\mathcal{A}_{\mathcal{R}}(\sigma)$ based on a random string $\mathcal{R}$ and at the end of the stream, outputs $\mathsf{out}(\mathcal{A})$. The goal is to optimize the space usage, i.e., the size of $\mathcal{A}_{\mathcal{R}}(\sigma)$. In this paper, we only consider algorithms that make a single pass over the input stream. We say that a streaming algorithm $\mathcal{A}$ has error δ if $\Pr_{\mathcal{R}}[\mathsf{out}(\mathcal{A}) \neq f(\sigma)] \leqslant \delta$. Further, we say that an algorithm $\mathcal{A}$ *solves* a function f if it has error $\leqslant 1/3$. The maximum number of bits stored by $\mathcal{A}$ over all possible inputs σ is denoted by $\text{Space}(\mathcal{A})$.

We define the *classical streaming complexity* of a function f as

$$\mathcal{S}(f) := \min\{\text{Space}(\mathcal{A}) : \mathcal{A} \text{ solves } f\}$$

Annotated Streaming. In the annotated streaming model of [CCM09], we have a space-bounded Verifier and an all-powerful Prover with unlimited space. Given an input stream σ, a *scheme* for computing a function $f(\sigma)$ is a triple $\mathcal{P} = (\mathcal{H}, \mathcal{A}, \mathsf{out})$, where $\mathcal{H}$ is a function that Prover uses to generate the help message or proof-stream $\mathcal{H}(\sigma)$ that she sends Verifier *after* the input stream, $\mathcal{A}$ is a data streaming algorithm that Verifier runs on σ using a random string $\mathcal{R}$ to produce a summary $\mathcal{A}_{\mathcal{R}}(\sigma)$, and out is an algorithm that Verifier uses to process the proof $\mathcal{H}(\sigma)$ and generate an output $\mathsf{out}(\mathcal{H}(\sigma), \mathcal{A}_{\mathcal{R}}(\sigma), \mathcal{R}) \in \text{range}(f) \cup \{\perp\}$, where $\perp$ denotes rejection of the proof. Note that if the proof length $|\mathcal{H}(\sigma)|$ is larger than the memory of the Verifier, then out is a streaming algorithm that processes $\mathcal{H}(\sigma)$ as a stream and stores a summary subject to its memory.

A scheme $\mathcal{P} = (\mathcal{H}, \mathcal{A}, \mathsf{out})$ has completeness error δ_c and soundness error δ_s if it satisfies

- (completeness) $\forall \sigma : \Pr_{\mathcal{R}}[\mathsf{out}(\mathcal{H}(\sigma), \mathcal{A}_R(\sigma), \mathcal{R}) \neq f(\sigma)] \leqslant \delta_c$;
- (soundness) $\forall \sigma, \mathcal{H}' : \Pr_{\mathcal{R}}[\mathsf{out}(\mathcal{H}', \mathcal{A}_{\mathcal{R}}(\sigma), \mathcal{R}) \notin \{f(\sigma), \perp\}] \leqslant \delta_s$.

We say that a scheme $\mathcal{P}$ *solves* a function f if $\mathcal{P}$ has $\delta_s, \delta_c \leqslant 1/3$ for computing f.

The *hcost* (short for "help cost") of a scheme $\mathcal{P} = (\mathcal{H}, \mathcal{A}, \mathsf{out})$ is defined as $\max_\sigma |\mathcal{H}(\sigma)|$, i.e., the maximum number of bits required to express a proof over all possible inputs σ. The *vcost* (short for "verification cost") is the maximum bits of space used by the algorithms $\mathcal{A}$ and out respectively, where the maximum is taken over all possible $(\sigma, \mathcal{R})$. The total cost $\mathsf{tcost}(\mathcal{P})$ is defined as the sum $\mathsf{hcost}(\mathcal{P}) + \mathsf{vcost}(\mathcal{P})$.

The *annotated streaming complexity* of a function f is defined as

$$\mathrm{AS}(f) := \min\{\mathsf{tcost}(\mathcal{P}) : \mathcal{P} \text{ solves } f\}.$$

Trivial Scheme. We say that a scheme $\mathcal{P}$ that solves a function f is *trivial* if $\mathsf{vcost}(\mathcal{P}) = \tilde{\Omega}(\mathcal{S}(f))$.

Definition 9.2.7 (Non-trivial Annotated Streaming complexity)**.** *The* non-trivial *annotated streaming complexity of a function f is*

$$\widehat{\mathrm{AS}}(f) := \min\{\mathit{tcost}(\mathcal{P}) : \mathcal{P} \textit{ is a non-trivial scheme that solves } f\}.$$

A scheme $\mathcal{P}$ with $\mathsf{hcost}(\mathcal{P}) = O(h)$ and $\mathsf{vcost}(\mathcal{P}) = O(v)$ is called an (h, v)-scheme. Again, if $\mathsf{hcost}(\mathcal{P}) = \widetilde{O}(h)$ and $\mathsf{vcost}(\mathcal{P}) = \widetilde{O}(v)$, we call $\mathcal{P}$ an $[h, v]$-scheme.

We remark that while we can prove statements analogous to Observation 9.2.4, with $\mathrm{MA}^{\rightarrow}(f)$, $\widehat{\mathrm{MA}}^{\rightarrow}(f)$, and $R^{\rightarrow}(f)$ replaced by $\mathrm{AS}(f)$, $\widehat{\mathrm{AS}}(f)$, and $\mathcal{S}(f)$ respectively, a statement analogous to Observation 9.2.3 does not follow. This is because unlike the communication setting, Prover sending the entire input to Verifier may not trivially solve the problem since Verifier still needs to process the proof using sublinear space.

9.2.2 Notation and Terminology

We define some notation and terminology that we use throughout the paper. For a string $z \in [\alpha]^d$, we use $z[k]$ to denote the element at the kth index of z. Although the standard notation to denote the input size in a communication problem is n, we use N instead to avoid confusion with the number of nodes (for which we use n).

All graphs in this paper are simple and undirected. For any two vertices $s, t \in V$, we say that a collection of s-t paths are vertex-disjoint if they do not share any vertices other than s and t and are edge disjoint if they do not share any edges.

9.3 Preliminaries

We recall a fundamental problem in communication complexity and state a basic fact about its one-way communication complexity, which we use multiple times in the paper.

Definition 9.3.1 (Equality)**.** *In the* EQ_N *problem, Alice and Bob hold strings* $x, y \in \{0.1\}^N$ *respectively, and their goal is to check whether* $x = y$*.*

Fact 9.3.2 ([KN97])**.** *For any* N*, there is an* $O(1)$*-cost public coin one-way protocol and an* $O(\log N)$*-cost private coin one-way protocol for* EQ_N *with error probability at most* $1/3$*.*

Fact 9.3.3 ([KN97])**.** *The one-way randomized communication complexity* $R^{\rightarrow}(\text{EQ}_N) = \Omega(\log N)$*.*

Fact 9.3.4 (Newman's Theorem [KN97])**.** *For any function* $f : \{0,1\}^N \times \{0,1\}^M \to \{0,1\}$*, a public coin protocol with communication cost* C *and error* ε *can be simulated by a private coin protocol with communication cost at most* $O(C + \log(N + M) + \log(1/\delta))$ *and error at most* $\varepsilon + \delta$*.*

We use a pseudorandom generator to reduce the space used by our random bits in certain algorithms. The fact that we exploit is given below.

Fact 9.3.5 (PRG [Nis90])**.** *Any randomized algorithm that runs in* S *space and uses one-way access to* R *random bits may be converted to an algorithm that uses* $O(S \log R)$ *random bits and runs in* $O(S \log R)$ *space using a pseudorandom generator. For dynamic streaming algorithms, the algorithm should be able to access the random bits in a read-once manner for some permutation of the stream.*

We often refer to the following generic lower bound in the OMA model as well as annotated streaming.

Fact 9.3.6 (General OMA lower bound, [CCMT14])**.** *For any function* f*, an* (h, v)*-OMA-procotol or an* (h, v)*-scheme* Π *solving it must have*

$$h \cdot v \geqslant \Omega(C)$$

where $C = R^{\rightarrow}(f)$ *in case* Π *is an OMA protocol, and* $C = \mathcal{S}(f)$ *(the streaming complexity of* f*) in case it is an annotated streaming scheme.*

The standard schemes below are often used as subroutines in our protocols.

Fact 9.3.7 (Subset-check and Intersection-count Scheme, [CCMT14], [CGT20])**.** *Given an insert-delete stream of elements from sets $X, Y \subseteq [N]$ (interleaved arbitrarily), for any h, v with $h \cdot v = N$, there are $[h, v]$-schemes for checking whether $X \subseteq Y$ and for counting $|X \cap Y|$.*

Fact 9.3.8 (Duplicate-detection Scheme)**.** *Given an insert-only stream of elements from the universe $[N]$, for any h, v with $h \cdot v = N$, there is an $[h, v]$-scheme for checking whether all elements in the stream are distinct.*

We now mention an important proposition for edge and vertex connectivity. The first is Menger's theorem which gives an equivalent definition of k-vertex-connectivity via vertex-disjoint paths.

Fact 9.3.9 (Menger's Theorem; c.f. [Wes01, Theorem 17])**.** *Let G be an undirected graph and s and t be two vertices. Then the size of the minimum edge cut for s and t is equal to the maximum number of pairwise edge-independent paths from s to t. If s, t are non-adjacent then the size of the minimum vertex cut for s and t is equal to the maximum number of vertex-disjoint paths between s and t.*
Moreover, a graph is k-edge-connected (resp k-vertex-connected) iff every pair of vertices has k edge-disjoint (resp vertex-disjoint) paths in between them.

We often use the following well-known sketch from the streaming literature.

Fact 9.3.10 ([AGM12a])**.** *There is an algorithm that given any n-vertex graph G in a dynamic stream computes a spanning forest T of G with probability at least $1 - n^{-10}$ in $O(n \log^3(n))$ bits of space.*

We give a new way to detect if two multi-sets are identical using ℓ_0-samplers. The idea is to insert elements of the first multi-set into an ℓ_0-sampler and delete elements of the second multi-set from the sampler, and at the end, check if the sampler is empty.

Proposition 9.3.11 (Equality-Detection Scheme)**.** *Given an insert-only stream of elements of two multi-sets A, B in any order (interleaved arbitrarily) from the universe $[N]$, we can check if the two sets are identical with probability $1 - 1/\text{poly}(N)$ in space $\text{polylog}(N)$.*

9.4 OMA Lower Bound for Equals-Index and its Implications

In this section, we prove our lower bound results. In Section 9.4.1, we introduce the problem EQ-IDX, which is at the core of our results, and prove a lower bound on its non-trivial OMA

complexity. Then in Section 9.4.2, we show applications of this lower bound in proving a collection of lower bounds in the OMA model as well as the annotated streaming model.

9.4.1 The Equals-Index Problem

Recall the EQ-IDX problem as defined in Definition 9.1.1. First, let us show tight bounds on its one-way randomized communication complexity.

Lemma 9.4.1. $R^{\rightarrow}(\text{EQ-IDX}_{p,q}) = \Theta(p + \log q)$

Proof. First consider the following protocol using public randomness. For each $i \in [p]$, Alice sends Bob an $O(1)$-size equality sketch (Fact 9.3.2) for x_i. Bob uses only the jth sketch to check if $x_j = y$. Thus, $R^{\rightarrow(\text{pub})}(\text{EQ-IDX}_{p,q}) = O(p)$. By Newman's theorem (Fact 9.3.4),

$$R^{\rightarrow}(\text{EQ-IDX}_{p,q}) = O(p + \log(pq)) = O(p + \log q)$$

The lower bound $R^{\rightarrow}(\text{EQ-IDX}_{p,q}) = \Omega(p + \log q)$ easily follows from the facts that $\text{EQ-IDX}_{p,1} \equiv \text{IDX}_p$ (Definition 2.3.2) and $\text{EQ-IDX}_{1,q} \equiv \text{EQ}_q$ (Definition 9.3.1). Therefore, we have

$R^{\rightarrow}(\text{EQ-IDX}_{p,q}) \geqslant R^{\rightarrow}(\text{EQ-IDX}_{p,1}) = R^{\rightarrow}(\text{IDX}_p) = \Omega(p)$ (Proposition 2.3.3).

Again, $R^{\rightarrow}(\text{EQ-IDX}_{p,q}) \geqslant R^{\rightarrow}(\text{EQ-IDX}_{1,q}) \geqslant \Omega(\log q)$ (Fact 9.3.3). ∎

Now we prove an OMA lower bound for EQ-IDX.

Lemma 9.4.2. *Any (h,v)-OMA-protocol solving* $\text{EQ-IDX}_{p,q}$ *must have*

$$(h + q) \cdot v \geqslant \Omega(pq)$$

Proof. We show that given any (h, v)-OMA-protocol Π for $\text{EQ-IDX}_{p,q}$, we can design an $(h + q, v)$-OMA-protocol Π' for IDX_{pq}. The lower bound of $(h + q) \cdot v \geqslant \Omega(pq)$ then immediately follows from the OMA lower bound for IDX_{pq} (Proposition 2.3.3).

Suppose Alice and Bob have inputs $x \in \{0,1\}^{pq}$ and $k \in [pq]$ in the IDX_{pq} problem. Then the protocol Π' is as follows. Alice partitions x into p chunks $x_1, \ldots, x_p$, each of size q. Bob sets $j = \lceil k/p \rceil$. Merlin sends $y \in \{0,1\}^q$ to Bob and claims that it equals x_j. The players can now interpret $(x_1, \ldots, x_p)$ and (y, j) as inputs to the $\text{EQ-IDX}_{p,q}$ problem, and run the protocol Π to verify that y is indeed equal to x_j. If the check passes, then Bob knows $x[k]$ since it lies in $x_j = y$. If not, he outputs $\perp$, i.e., rejects the proof.

We analyze the completeness and soundness errors of Π'. If Merlin is honest, then y is indeed x_j, and the protocol fails only if Bob rejects because the check in Π doesn't go through. The probability of this is exactly the completeness error of Π. Hence, Π' has the same completeness error as Π. Again, if Merlin is dishonest, then the protocol fails only when Bob outputs the incorrect bit-value for $x[k]$. This happens only when $y \neq x_j$, but the check passes in Π. This has the same probability as the soundness error of Π. Therefore, Π' also has the same soundness error as Π. Thus, by definition, since Π solves $\textsc{eq-idx}_{p,q}$, the protocol Π' solves $\textsc{idx}_p$.

Finally, we analyze the cost of Π'. Merlin sends Bob y as well as his message due to Π. Thus, $\mathrm{hcost}(\Pi') = O(h+q)$. Alice sends Bob only her message due to Π, implying that $\mathrm{vcost}(\Pi') = v$. Thus, Π' is an $(h+q, v)$-OMA-protocol as claimed. This completes the proof. ∎

Our main result on the non-trivial OMA complexity $\widehat{\mathrm{MA}}^{\rightarrow}(\textsc{eq-idx}_{p,q})$ follows from the above two lemmas.

Theorem 17. *For any p, q with $p = \Omega(\log q)$, we have $\widehat{\mathrm{MA}}^{\rightarrow}(\textsc{eq-idx}_{p,q}) = \omega(q)$*

Proof. By Lemma 9.4.1, we have $R^{\rightarrow}(\textsc{eq-idx}_{p,q}) = \Theta(p)$ for this setting of p and q. Therefore, any non-trivial OMA protocol Π for the problem must have $\mathsf{vcost}(\Pi) = o(p)$. But by Lemma 9.4.2, if $\mathsf{hcost}(\Pi) = O(q)$, then $\mathsf{vcost}(\Pi) = \Omega(p)$. Therefore, $\mathsf{hcost}(\Pi)$ must be $\omega(q)$, which means $\mathsf{tcost}(\Pi) = \omega(q)$. Then, by definition, $\widehat{\mathrm{MA}}^{\rightarrow}(\textsc{eq-idx}_{p,q}) = \omega(q)$. ∎

Corollary 9.1.2. *There is an explicit function f with*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) = \exp(\Omega(\mathrm{MA}^{\rightarrow}(f))) \text{ and } \widehat{\mathrm{MA}}^{\rightarrow}(f) = \exp(\Omega(R^{\rightarrow}(f))).$$

Proof. Set $f = \textsc{eq-idx}_{p,q}$ with $p = \log N$ and $q = N/\log N$. Indeed, $p \geqslant \log q$. Hence, by Theorem 17, we have $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(q) = \exp(\Omega(p))$.

By Lemma 9.4.1, we have $R^{\rightarrow}(f) = \Theta(p)$. Also, by definition $\mathrm{MA}^{\rightarrow}(f) \leqslant R^{\rightarrow}(f) = \Theta(p)$. Therefore, the claimed result holds. ∎

Corollary 9.1.3. *For any $C \in (\sqrt{N}, N/\log N)$, there is an explicit function f on input size N with*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(C).$$

Proof. Set $f = \textsc{eq-idx}_{p,q}$ with $p = N/C$ and $q = C$. Since $q = C \leqslant N/\log N$, we have

$$p = N/C \geqslant \log N > \log q$$

By Theorem 17, $\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(C)$. ∎

We also give a tight bound on the (standard) OMA complexity of EQ-IDX$_{p,q}$.

Theorem 25. *For any p, q with $p \geqslant \log q$, we have* $\mathrm{MA}^{\rightarrow}(\text{EQ-IDX}_{p,q}) = \Theta(\min\{\sqrt{pq}, p\})$.

Proof. First, we prove the lower bound. Given an (h, v)-OMA-protocol for EQ-IDX$_{p,q}$, first consider the case that $h > q$. Then, by Lemma 9.4.2, we have $h \cdot v = \Omega(pq)$, which means $h + v = \Omega(\sqrt{pq})$. Otherwise, i.e., if $h \leqslant q$, then by the same lemma, we have $q \cdot v = \Omega(pq)$, which means $v = \Omega(p)$, and hence, $h + v = \Omega(p)$. Therefore, we conclude

$$\mathrm{MA}^{\rightarrow}(\text{EQ-IDX}_{p,q}) = \Omega(\min\{\sqrt{pq}, p\}) \tag{9.1}$$

For the upper bound, we first show that if $q \leqslant p$, then we can design a $(\sqrt{pq}, \sqrt{pq})$-OMA-protocol for EQ-IDX$_{p,q}$. Alice combines $x_1, \ldots, x_p$ into a single pq-bit string x, and then again splits x into $\sqrt{pq}$ chunks $z_1, \ldots, z_{\sqrt{pq}}$, each of which has $\sqrt{pq}$ bits. Observe that since $q \leqslant p$, we have $q \leqslant \sqrt{pq}$, and hence, x_j lies entirely in (the concatenation of) at most two consecutive chunks $z_k \circ z_{k+1}$. Merlin sends Bob z_k and z_{k+1} that he claims are identical to z'_k and z'_{k+1}. This takes $O(\sqrt{pq})$ bits. If they are indeed as claimed, Bob can determine whether $y = x_j$. The problem now reduces to checking whether $z'_k = z_k$ and $z'_{k+1} = z_{k+1}$ without Merlin. To this end, it suffices to show that there exists a one-way randomized private-coin protocol of cost $O(\sqrt{pq})$. The protocol is very similar to the one mentioned in the proof of Lemma 9.4.1. An $O(\sqrt{pq})$ public-coin protocol can be obtained by having Alice send an $O(1)$-size equality sketch (Fact 9.3.2) for each z_i, while Bob uses only z_k and z_{k+1} for the check. Newman's theorem (Fact 9.3.4) now gives a private-coin protocol of cost $O(\sqrt{pq} + \log(pq)) = O(\sqrt{pq})$. Therefore, we obtain an OMA protocol of total cost $O(\sqrt{pq})$ for $q \leqslant p$.

Finally, note that $\mathrm{MA}^{\rightarrow}(\text{EQ-IDX}_{p,q})$ is always trivially upper bounded by $R^{\rightarrow}(\text{EQ-IDX}_{p,q})$, which, by Lemma 9.4.1, is $O(p)$ for this setting of $p \geqslant \log q$. Hence, we can conclude that whenever $p \geqslant \log q$,

$$\mathrm{MA}^{\rightarrow}(\text{EQ-IDX}_{p,q}) = O(\min\{\sqrt{pq}, p\}). \tag{9.2}$$

Thus, the theorem follows from eq (9.1) and eq (9.2). ∎

9.4.2 Implications of the Equals-Index Lower Bounds

Here, we show how the EQ-IDX lower bound can be used to prove multiple interesting lower bounds in the OMA and annotated streaming models.

Sparse Index

Consider the SP-IDX$_{M,N}$ problem as described in Section 9.1.2. This is the IDX$_N$ problem (Definition 2.3.2) with the promise that Alice's string has hamming weight at most M. First, we prove a tight bound on its randomized one-way communication complexity. This tells us the benchmark for vcost of a non-trivial OMA protocol for the problem. It also improves upon the previously best-known upper bound of $O(M \log M)$ proven by [CCMT14].

Lemma 9.4.3. $R^{\rightarrow}(\text{SP-IDX}_{M,N}) = \Theta(M + \log\log N)$

Proof. First, we describe a public-coin protocol for SP-IDX$_{M,N}$ with cost $O(M)$. Using public randomness, Alice and Bob sample a function $h : [N] \to [3M]$ from a pairwise independent hash family $\mathcal{H}$. Let $Y := \{i \in [N] : x[i] = 1\}$. Alice sends Bob the $3M$-size characteristic vector of the set $S = \{h(i) : i \in Y\}$. Bob checks whether $h(j) \in S$. If yes, he announces $x[j] = 1$, and otherwise he announces $x[j] = 0$. Observe that if $x[j]$ is indeed 1, then $h(j) \in S$, and Bob is always correct. Otherwise, if $x[j]$ is actually 0, Bob errs only when $h(j) = h(i)$ for some $i \in Y$. Fix such an i. By pairwise independence of $\mathcal{H}$, we have

$$\Pr_{h \in \mathcal{H}}[h(j) = h(i)] = \frac{1}{3M}.$$

By union bound over all $i \in Y$, and since $|Y| \leqslant M$ by definition of SP-IDX$_{M,N}$, we get

$$\Pr_{h \in \mathcal{H}}[\exists i \in Y : h(j) = h(i)] \leqslant \frac{1}{3}.$$

Thus, the error of the protocol is at most $1/3$. Therefore, we get $R^{\rightarrow(\text{pub})}(\text{SP-IDX}_{M,N}) = O(M)$.

To get a private coin protocol, we appeal to Newman's theorem (Fact 9.3.4) once again. To obtain a tight bound, we need to have an appropriate interpretation of the problem. Observe that since the number of possible inputs to Alice is $\binom{N}{M}$, we can interpret the domain of SP-IDX$_{M,N}$ as $\{0,1\}^{\log\binom{N}{M}} \times \{0,1\}^{\log N}$. Thus, by Fact 9.3.4, we get

$$R^{\rightarrow}(\text{SP-IDX}_{M,N}) = O\left(M + \log\log\binom{N}{M}\right) = O(M + \log M + \log\log N) = O(M + \log\log N)$$

For the lower bound part, an $\Omega(M)$ lower bound follows trivially by reduction from IDX$_M$: given her input $x \in \{0,1\}^M$, Alice appends it with $N - M$ 0's to its right to turn

it into an N-bit string with at most M 1's. Given Bob's index $j \in [M]$, the jth index of the resultant string is the same as that of the original string. Then, we can use a protocol for $\text{SP-IDX}_{N,M}$ to solve IDX_M.

Again, an $\Omega(\log \log N)$ lower bound follows (by Fact 9.3.3) since $R^{\rightarrow}(\text{SP-IDX}_{1,N})$ can be reduced from $\text{EQ}_{\log N}$: given her input $x \in \{0,1\}^{\log N}$, Alice sees x as a number in $[N]$, and constructs an N-bit string with only the xth index having a 1 and all other indices having 0. Bob looks at his input $y \in \{0,1\}^{\log N}$ as a number in $[N]$ and sets his index in the $\text{SP-IDX}_{1,N}$ problem as y. The output to this instance of $\text{SP-IDX}_{M.N}$ is 1 iff $x = y$. Hence, we also get

$$R^{\rightarrow}(\text{SP-IDX}_{M,N}) = \Omega(\max\{M, \log \log N\}) = \Omega(M + \log \log N).$$

This completes the proof. ∎

Our next lemma proves a lower bound for any OMA protocol that solves SP-IDX. We remark that the proof is similar to the one by [CCMT14], but they reduce from IDX, while we reduce from EQ-IDX. This lets us improve upon the previous lower bound.

Lemma 9.4.4. *Any (h, v)-OMA-protocol for $\text{SP-IDX}_{M,N}$ must have*

$$(h + \log(N/M)) \cdot v \geqslant \Omega(M \log(N/M))$$

Proof. We reduce from $\text{EQ-IDX}_{p,q}$. Given any instance of this problem, we construct an instance of $\text{SP-IDX}_{M,N}$ where $M = p$ and $N = 2^q p$ (hence, $q = \log(N/M)$) such that an (h, v)-OMA-protocol that solves the $\text{SP-IDX}_{M,N}$ instance also solves the $\text{EQ-IDX}_{p,q}$ instance (without any additional hcost or vcost). The lower bound then follows from Lemma 9.4.2.

Recall that Alice's input to the $\text{EQ-IDX}_{p,q}$ problem is $(x_1, \ldots, x_p)$, where each $x_i \in \{0,1\}^q$. Alice interprets each x_i as the binary representation of an integer in $[0, 2^q - 1]$. She then constructs a string z of length $N = 2^q p$ indexed by pairs (i, j) where $i \in [p]$ and $j \in [0, 2^q - 1]$. For each $i \in [p]$, she sets $z[i, x_i] = 1$. She sets all the other bits of z to 0. Since z has exactly p 1's, its hamming weight is $M = p$. Again, Bob, who has inputs $y \in \{0,1\}^q$ and $j \in [p]$ to $\text{EQ-IDX}_{p,q}$, sets $k := (j, y)$ interpreting y as an integer in $[0, 2^q - 1]$. Alice and Bob now solve $\text{SP-IDX}_{M,N}$ with the inputs z and k respectively. Indeed, by construction, $z[k] = 1$ if and only if $y = x_j$. Therefore, an (h, v)-OMA-protocol that solves $\text{SP-IDX}_{M,N}$ can be used to solve any instance of $\text{EQ-IDX}_{p,q}$ with $p = M$ and $q = \log(N/M)$. ∎

Note that our bound is strictly better than Theorem 3.9 of [CCMT14], which (essentially) says that an (h, v)-OMA-protocol for $\text{SP-IDX}_{M,N}$ must have $(h + \log N) \cdot v \geqslant \Omega(M \log(N/M))$; this inequality is trivially implied by Lemma 9.4.4. Further, our strengthening of the additive factor from $\log N$ to $\log(N/M)$, combined with our tight classical one-way communication bound of SP-IDX, enables us to prove strong lower bounds on its non-trivial OMA complexity as we show below.

Consider $\text{SP-IDX}_{M,N}$ for $M \geqslant \log\log N$. By Lemma 9.4.3, its one-way randomized communication complexity is $\Theta(M)$. Therefore, any non-trivial OMA protocol for the problem must have vcost $o(M)$. But by Lemma 9.4.4, for any (h, v)-OMA-protocol for SP-IDX,

$$h = O(\log(N/M)) \Rightarrow v = \Omega(M)$$

Hence, any non-trivial OMA-protocol Π must have $\mathsf{hcost}(\Pi) = \omega(\log(N/M))$, which means whenever $M \geqslant \log\log N$, we have $\widehat{\text{MA}}^{\rightarrow}(\text{SP-IDX}_{M,N}) = \omega(\log(N/M))$. This establishes Theorem 18.

Theorem 18. *For $M = \log\log N$, we have $\widehat{\text{MA}}^{\rightarrow}(\text{SP-IDX}_{M,N}) = \omega(\log N)$, whereas $R^{\rightarrow}(\text{SP-IDX}_{M,N}) = \text{MA}^{\rightarrow}(\text{SP-IDX}_{M,N}) = \Theta(\log\log N)$.*

Proof. The bound of $\widehat{\text{MA}}^{\rightarrow}(\text{SP-IDX}_{M,N})$ follows from above since for $M = \log\log N$, we have $\log(N/M) = \Theta(\log N)$. The bound of $R^{\rightarrow}(\text{SP-IDX}_{M,N})$ follows by Lemma 9.4.3. Finally, Lemma 9.4.4 implies that any (h, v)-OMA-protocol for $\text{SP-IDX}_{\log\log N,N}$ must either have $h = \Omega(\log N)$ or $v = \Omega(\log\log N)$. Thus, in any case, $h + v \geqslant \Omega(\log\log N)$, i.e., $\text{MA}^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) = \Omega(\log\log N)$. Again, by definition, $\text{MA}^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) \leqslant R^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) = O(\log\log N)$. Therefore, we conclude $\text{MA}^{\rightarrow}(\text{SP-IDX}_{\log\log N,N}) = \Theta(\log\log N)$. ∎

Hence, we show an exponential separation between $\widehat{\text{MA}}^{\rightarrow}$ complexity and $R^{\rightarrow}$ (or $\text{MA}^{\rightarrow}$) complexity of the Sparse Index problem with sparsity $\log\log N$.

Connectivity and Bipartite Testing

Now we show that by reduction from EQ-IDX, we can prove $\widehat{\text{MA}}^{\rightarrow}$ lower bounds for the only two other functions that were known to have $\text{MA}^{\rightarrow}$ complexity—and implicitly $\widehat{\text{MA}}^{\rightarrow}$ complexity—nearly as large as their $R^{\rightarrow}$ complexity. These are XOR-Connectivity (XOR-CONN) and XOR-Bipartiteness (XOR-BIP), as studied in [Tha16b]. For two sets $S, T \subseteq \mathcal{U}$, we define their XOR $S \oplus T$ as their symmetric difference $(S \cup T) \setminus (S \cap T)$. We are now ready to formally define the problems below.

XOR-CONN$_n$: Alice and Bob have graphs $G_A = (V, E_A)$ and $G_B = (V, E_B)$ on the same set of nodes V with $|V| = n$. The graph G is defined as $(V, E_A \oplus E_B)$. The goal is to determine whether G is connected: Bob needs to output 1 if it is and 0 otherwise.

XOR-BIP$_n$: Alice and Bob have graphs $G_A = (V, E_A)$ and $G_B = (V, E_B)$ on the same set of nodes V with $|V| = n$. The goal is to determine whether $G := (V, E_A \oplus E_B)$ is bipartite: Bob needs to output 1 if it is, and 0 otherwise.

As noted by [Tha16b], one can use a modified version of the connectivity- and bipartiteness-sketches by [AGM12a] to obtain $\widetilde{O}(n)$-space classical streaming algorithms for the *streaming versions* of the XOR-CONN$_n$ and XOR-BIP$_n$ problems respectively: in these versions, we need to solve connectivity and bipartiteness in an "XOR-update" streaming model, where an edge insertion deletes the edge if it already exists in the graph, and adds it to the graph otherwise. These classical streaming algorithms, by the standard streaming-to-communication simulation, give $\widetilde{O}(n)$-cost one-way randomized protocols for XOR-CONN$_n$ and XOR-BIP$_n$ in the classical communication model. This means $R^{\rightarrow}$(XOR-CONN$_n$) and $R^{\rightarrow}$(XOR-BIP$_n$) are both $\widetilde{O}(n)$. Well-known communication lower bounds for (standard) connectivity [FKM+04,SW15] and bipartiteness [FKM+04] also imply that $R^{\rightarrow}$(XOR-CONN$_n$) and $R^{\rightarrow}$(XOR-BIP$_n$) are actually $\tilde{\Theta}(n)$.

[Tha16b] showed that MA$^{\rightarrow}$(XOR-CONN$_n$) and MA$^{\rightarrow}$(XOR-BIP$_n$) are $\Omega(n)$. This follows from their more general result:

Theorem 26 ([Tha16b], Theorem 4.1, restated)**.** *Any (h, v)-OMA-protocol solving* XOR-CONN$_n$ *or* XOR-BIP$_n$ *must have*

$$(h + \Theta(n)) \cdot v \geqslant \Omega\left(n^2\right).$$

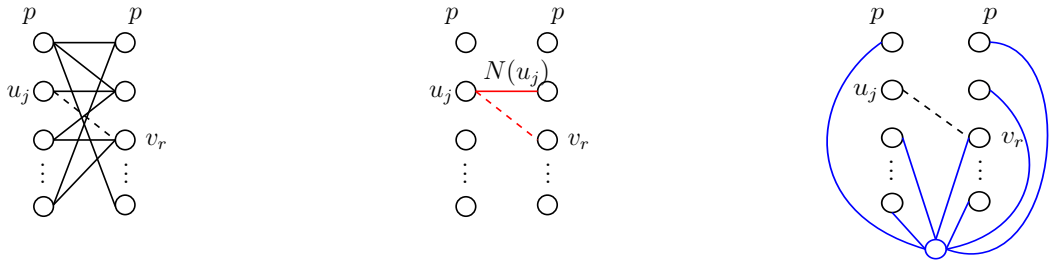
Since a non-trivial OMA protocol for the problems must attain $\mathsf{vcost} = o(n)$, we get the following corollary.

Corollary 9.1.4. *([Tha16b], paraphrased) For $f \equiv$* XOR-CONN$_n$ *or $f \equiv$* XOR-BIP$_n$*, we have*

$$\widehat{\mathrm{MA}}^{\rightarrow}(f) = \omega(R^{\rightarrow}(f)) = \omega(n).$$

We prove that Theorem 26 also follows by reduction from EQ-IDX$_{p,q}$ for appropriate settings of p and q, thus adding to the usefulness of EQ-IDX in proving strong $\widehat{\mathrm{MA}}^{\rightarrow}$ lower bounds.

Proof of Theorem 26. First, we prove the result for XOR-CONN$_n$. The construction for XOR-BIP$_n$ is very similar, and so we simply mention the modifications for that case.



(a) Alice uses x_i to construct the neighborhood of vertex i.

(b) Bob constructs $N(u_j)$ based on y.

(c) Bob creates an additional vertex and connects all vertices except u_j to it.

Figure 9.1: Construction for XOR-CONN$_n$

WLOG, let n be odd. Let $p := (n-1)/2$. We reduce from EQ-IDX$_{p,p}$. Alice sees her input $(x_1, \ldots, x_p)$, where each $x_i \in \{0,1\}^p$, and constructs a graph on the vertex set $V := V_L \cup V_R \cup \{s\}$, where $V_L = \{u_1, \ldots, u_p\}$ and $V_R = \{v_1, \ldots, v_p\}$ are the partite sets of the bipartite graph, and s is a dummy node outside it. Clearly, $|V| = n$. For each $\ell \in [p]$, Alice joins the edges $\{\{u_\ell, v_r\} : x_\ell[r] = 1, r \in [p]\}$. This completes the description of the graph $G_A = (V, E_A)$ where E_A is the set of all edges added by Alice. Again, given his inputs $y \in \{0,1\}^p$ and $j \in [p]$, Bob joins the edges $\{\{u_j, v_r\} : y[r] = 1, r \in [p]\}$. He then joins vertex s to all nodes in $V_L \cup V_R$ except u_j. This completes the description of the graph $G_B = (V, E_B)$ where E_B is the set of all edges added by Bob.

Observe that if y is indeed equal to x_j, then the neighborhoods of the vertex u_j in the graphs G_A and G_B are identical by construction. Then, u_j is an isolated vertex in $G = (V, E_A \oplus E_B)$, which means G is not connected in this case. Otherwise, if $y \neq x_j$, then there must exist $r \in [p]$ such that $y[r] \neq x_j[r]$. Thus, by construction, exactly one of the sets E_A and E_B has the edge $e := \{u_j, v_r\}$, and hence $e \in E_A \oplus E_B$. Therefore, in G, the node u_j is connected to v_r, which, in turn, is connected to s. Observe that every node $w \in V_L \cup V_R$ with $w \neq u_j$ is connected to s in G: $\{s, w\}$ exists in E_B but not in E_A. Thus, s is connected to every other node in G, implying that G is connected. Hence, G is connected iff $y \neq x_j$. Therefore, given an (h, v)-OMA-protocol for XOR-CONN$_n$, the players can run it for the constructed graph G, and Bob can determine the output to EQ-IDX$_{p,p}$ with $p = (n-1)/2$. Lemma 9.4.2 now implies the theorem for XOR-CONN$_n$.

For XOR-BIP$_n$, the construction of the graphs G_A and G_B are the same, *except* Bob joins s only to u_j and all vertices in V_R (instead of $V_L \cup V_R \setminus \{u_j\}$). If $x_j = y$, then the neighborhoods of u_j in G_A and G_B are the same, except u_j is also adjacent to s in G_B. Hence, in G, the only edge incident on u_j is $\{u_j, s\}$. Therefore, we can bipartition G into

$(\{s\} \cup V_L \setminus \{u_j\}, V_R \cup \{u_j\})$. Observe that this is a valid bipartition since each partite set is an independent set in G. Otherwise, if $x_j \neq y$, then by the same logic as in the case of XOR-CONN$_n$, there must exist $r \in [p]$ such that $\{u_j, v_r\} \in G$. Further, $\{s, v_r\}, \{s, u_j\} \in G$ since these edges are in E_B but not in E_A. Hence, G has a triangle in this case and is not bipartite. Thus, we conclude that G is bipartite iff the answer to $x_j = y$, which means an (h, v)-OMA-protocol for XOR-BIP$_n$ can be used to determine the solution to any instance of EQ-IDX$_{p,p}$ with $p = (n-1)/2$. The claim again follows from Lemma 9.4.2.

Distinct Items

Now we turn to annotated streaming and discuss lower bounds for the well-studied distinct items problem under turnstile streams (see Section 9.2.1). We formally define the problem as follows.

DIST-ITEM$_{N,F}$: Given a turnstile stream of elements in $[N]$ such that $\max\{|\text{freq}(j)| : j \in [N]\} \leqslant F$, output the number of items with non-zero frequency, i.e., $|\{j \in [N] : \text{freq}(j) \neq 0\}|$

Lemma 9.4.5. *Any (h, v)-scheme for* DIST-ITEM$_{N,F}$ *must have*

$$(h + \log F) \cdot v \geqslant \Omega(N \log F)$$

Proof. Given an (h, v)-scheme $\mathcal{A}$ for DIST-ITEM$_{N,F}$, we design an (h, v)-OMA-protocol for EQ-IDX$_{N,\log F}$. In an instance of this problem, Alice has $x_1, \ldots, x_N$, where each $x_i \in \{0,1\}^{\log F}$. She interprets each x_i as (the binary representation of) an integer in $[0, F-1]$ and feeds $x_i + 1$ insertions of the element i to $\mathcal{A}$. Observe that $1 \leqslant \text{freq}(i) \leqslant F$ for each i in the stream. Alice then sends Bob the memory state of $\mathcal{A}$. Bob has inputs $y \in \{0,1\}^{\log F}$ and $j \in [N]$. Again, he interprets y as an integer in $[0, F-1]$ and continues the run of $\mathcal{A}$ by appending the stream with $y + 1$ deletions of the item j. Merlin sends Bob the proof that the Prover would send in $\mathcal{A}$. Bob can now run the verification part of $\mathcal{A}$ and get an output. If the answer is $n - 1$, then he announces that $x_j = y$. Otherwise, he declares $x_j \neq y$. For the correctness, observe that Alice added the item j to the stream $x_j + 1$ times and Bob deleted it $y + 1$ times. Therefore, $\text{freq}(j) = 0$ iff $x_j = y$. All other stream elements have frequency at least 1. Hence, the number of items with non-zero frequency in the stream is $N - 1$ if $x_j = y$, and N otherwise. ∎

Again, we need to prove a tight bound on the classical streaming complexity of DIST-ITEM so that we know what it means to be a non-trivial scheme for the problem. Using the non-zero detectors that we build in Section 9.5.1, we can design an efficient classical streaming algorithm for DIST-ITEM$_{N,F}$ for large F.

Lemma 9.4.6. *There exists a classical streaming algorithm using $O(N \log\log F)$ space for* DIST-ITEM$_{N,F}$. *Again, any such algorithm needs $\Omega(N)$ space.*

Proof. We can keep one non-zero detector given by Corollary 9.5.3 for each element in $[N]$, and can figure out the answer. by Corollary 9.5.3, we need $\widetilde{O}(N \log\log F)$ space and this succeeds with probability $1 - 1/\text{polylog}\,(F)$ by union bound over the N detectors. The lower bound follows by a simple reduction from IDX$_N$, even for insert-only streams: Alice inserts $\{i : x_i = 1\}$ and sends Bob the hamming weight of x. Bob inserts j and checks whether or not the number of distinct elements in the stream exceeds the hamming weight of x. If yes, $x_j = 0$, and if not x_j must be 1. ∎

Given the above lemma, we can prove our lower bound on the non-trivial total cost of DIST-ITEM$_{n,F}$.

Lemma 9.4.7. *Any non-trivial scheme for* DIST-ITEM$_{N,F}$ *with $F \leqslant \exp(N^{\text{polylog}\,(N)})$ must have total cost $\omega(\log F)$.*

Proof. By Lemma 9.4.6, the classical streaming complexity of DIST-ITEM$_{N,F}$ is $\tilde{\Theta}(N)$ for $F \leqslant \exp(N^{\tilde{\Theta}(1)})$. Hence, any non-trivial scheme Π for the problem must have $\mathsf{vcost}(\Pi) = o(N)$. By Lemma 9.4.5, $\mathsf{hcost}(\Pi) = O(\log F)$ implies $\mathsf{vcost}(\Pi) = \Omega(N)$. Hence, $\mathsf{hcost}(\Pi) = \omega(\log F)$, which means $\mathsf{tcost}(\Pi)$ must be $\omega(\log F)$. ∎

This establishes a separation between the classical streaming complexity and the (non-trivial) annotated streaming complexity of DIST-ITEM and proves Theorem 19.

Theorem 19. *There is a setting of F such that* DIST-ITEM$_{N,F}$ *can be solved in $\widetilde{O}(N)$ space in classical streaming, but any non-trivial annotated streaming scheme for the problem must have total cost $\Omega(N^{\text{polylog}\,(N)})$.*

Connectivity Problems in Support Graph Turnstile Streams

We first make the following claim about classical streaming complexities of connectivity and k-connectivity.

Claim 9.4.8. *In the classical streaming model under SGT streams, graph connectivity can be solved in $\widetilde{O}(n)$ space, and each of the k-vertex-connectivity and k-edge-connectivity problems can be solved in $\widetilde{O}(kn)$ space. These bounds match (up to polylogarithmic factors) the space-bound known for each problem under dynamic graph streams.*

Proof. The upper bounds of connectivity and k-vertex-connectivity follow by using the [AGM12a] and [AS23a] algorithms respectively, and replacing their ℓ_0 samplers with our strong ℓ_0 samplers given by Lemma 9.5.4. For the k-edge connectivity problem, it is not clear that the algorithm by [AGM12a] can be implemented under SGT streams. So we design a new $\widetilde{O}(kn)$-space k-edge-connectivity algorithm under such streams, presented in Section 9.5.2. The claim then follows. ∎

We show a lower bound for connectivity in this model. First we prove the following lemma.

Lemma 9.4.9. *Any (h, v)-scheme for connectivity satisfies $(h+qn)\cdot v \geqslant n^2 \cdot q$. In particular, when $q = n^{\text{polylog}\,(n)}$, for $v = o(n)$ we need $h \geqslant n^{\text{polylog}\,(n)}$.*

The proof of this lemma is along the same lines as the proof of XOR-Connectivity except that we use $\text{EQ-IDX}_{n,qn}$. Alice and Bob construct the same graph, but now for each pair of vertices there could be multiple edge insertions (up to 2^q).

We now prove lower bounds for k-vertex-connectivity and k-edge-connectivity in this model.

Lemma 9.4.10. *Any (h, v)-scheme for k-vertex-connectivity or k-edge-connectivity satisfies $(h + q) \cdot v \geqslant kn \cdot q$. In particular, when $q = 2^{\text{polylog}\,(n)}$, for $v = o(kn)$ we need $h \geqslant 2^{\text{polylog}\,(n)}$.*

Say that we are given an (h, v)-scheme for k-vertex-connectivity or k-edge-connectivity in the Support Graph Turnstile streaming model. Given an instance of $\text{EQ-IDX}_{kn,q}$, Alice constructs the following graph stream. Let $V_L = \{u_1, \ldots, u_n\}$ and $V_R = \{v_1, \ldots, v_k\}$. For each $\ell \in [kn]$, Alice inserts an edge between the ℓ^{th} pair of vertices x_ℓ times. Given his inputs $y \in \{0, 1\}^q$ and $j \in [kn]$, Bob creates the following graph stream. Bob deletes an edge between the j^{th} pair of vertices y times. Bob also adds an edge between every pair of vertices except the j^{th} pair. Alice runs the (h, v)-scheme on her graph stream and sends the memory content to Bob, who then continues running the scheme on his graph stream. Iff the graph is k-connected, then output 0 for the $\text{EQ-IDX}_{kn,q}$ instance and output 1 otherwise. Note that the complete bipartite graph is k-connected, but even if you remove one edge it is not k-connected. The correctness follows from the following claim.

Claim 9.4.11. *The graph formed by the stream is k-connected in the support graph turnstile model iff the $\text{EQ-IDX}_{kn,q}$ instance has value 0.*

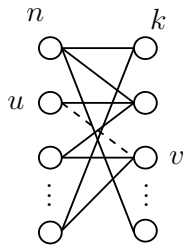
Proof. Consider the case when the EQ-IDX$_{kn,q}$ instance has value 1. We have $x_j = y$, so the j^{th} pair of vertices has y edge insertions and y edge deletions. This means that there is no edge between the j^{th} pair of vertices, implying that the support graph is not k-connected.

Now consider the case when the EQ-IDX$_{kn,q}$ instance has value 0. We have $x_j \neq y$, so the j^{th} pair of vertices has a different number of edge insertions and edge deletions. This implies that the j^{th} pair of vertices has a non-zero support implying that the j^{th} edge exists. Every other pair of vertices has only edge insertions, including exactly 1 edge insertion by Bob. This implies that every other pair of vertices has a non-zero support implying that the edge exists. Thus, the support graph is a complete bipartite graph implying that the support graph is k-connected. ∎

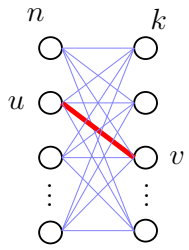
Claim 9.4.11 shows that this protocol solves EQ-IDX$_{kn,q}$. We know by Lemma 9.4.2 that the following holds for any protocol that solves EQ-IDX$_{kn,q}$:

$$(h+q) \cdot v \geqslant kn \cdot q.$$

The only help from Merlin is h bits for the (h, v)-scheme. The only communication from Alice is sending the memory content that takes v bits of space. This bound implies that even if $h = O(q)$, $v = \Omega(kn)$. So for $v = o(kn)$ we need $h = \omega(q)$. Setting $q = 2^{\text{polylog}\,(n)}$ implies that for $v = o(kn)$ we need $h \geqslant 2^{\text{polylog}\,(n)}$. However, using the strong ℓ_0-Samplers gives us $h = 0$ and $v = \widetilde{O}(kn)$.



(a) Alice adds x_i edges for pair i.



(b) Bob deletes (u, v) y times and adds every other pair once.

9.5 Classical Streaming Algorithms under SGT streams

9.5.1 Basic Tools: Non-Zero Detection and ℓ_0-Sampling for Large Frequencies

In this section, we introduce an ℓ_0-sampler that works when the entries in the vector are exponential in size.

Non-zero detection

Consider the following simple problem.

Problem 8. *Given a stream containing $+$'s and $-$'s output whether the number of $+$'s is exactly equal to the number of $-$'s or not. The promise is that the difference between the number of $+$'s and $-$'s is less than α at the end of the stream.*

A simple solution to Problem 8 is to store a deterministic counter mod α which takes $O(\log \alpha)$ bits of space. It is easy to show that we cannot do better deterministically.

Claim 9.5.1. *Any deterministic algorithm for Problem 8 needs $\log \alpha$ bits of space.*

Proof. Assume towards a contradiction that there is a deterministic algorithm that solves Problem 8 in less than $\log \alpha$ bits of space. Consider the different states of memory when we have exactly i $+$'s in the input stream. Since the memory used is less than $\log \alpha$ bits, there exist i and j (wlog $i < j$) such that the state of memory after inserting i times is the same as the state of memory after inserting j times. Now consider i $-$'s appear in the stream after the $+$'s. In the case where there were i $+$'s, the answer is 0, but the answer in the other case is 1. However, the algorithm cannot differentiate between the two cases because the memory state was the same after seeing the $+$'s, giving us a contradiction. ∎

Now the question is whether randomized algorithms can do better. Claim 9.5.1 implies a lower bound of $\Omega(\log \log \alpha)$ bits for the private randomness communication complexity of this problem. There exists a randomized algorithm for this problem that uses $O(\log \log \alpha)$ bits of space.

Lemma 9.5.2. *There exists a randomized algorithm that solves Problem 8 with probability $1 - 1/\text{polylog}(\alpha)$ and uses $O(\log \log \alpha)$ bits of space.*

Proof. Let p be a random prime within the first $\text{polylog}(\alpha)$ primes. Storing p takes $O(\log \log \alpha)$ bits of space by the prime density theorem. We store a counter mod p to solve Problem 8.

Let the counter value be $\alpha' < \alpha$. We make a mistake if our counter is 0, which happens when p is a factor of α'. Observe that α' has at most $\log \alpha$ prime factors because the product of all prime factors is at most $\alpha' < \alpha$ and all prime factors are at least 2.

So the probability that p divides α' is at most $\log \alpha / \text{polylog}(\alpha)$ implying that the probability of success is at least $1 - 1/\text{polylog}(\alpha)$. ∎

If we just slightly change the proof and let p be a random prime within the first $\text{polylog}(\alpha) \cdot \text{poly}(n)$ primes, then we get the following:

Corollary 9.5.3. *There exists a randomized algorithm that, given a parameter n, solves Problem 8 with probability $1 - 1/\text{polylog}(\alpha) \cdot \text{poly}(n)$ and uses $O(\log\log\alpha + \log n)$ bits of space.*

Strong ℓ_0-samplers

In this subsection, we design an ℓ_0-sampler that can handle large values. Let the universe of elements be $[n]$. Consider the following problem:

Problem 9. *Given a stream containing insertions and deletions of elements in $[n]$, output an element whose frequency is non-zero. The promise is that the value of each coordinate is between $-\alpha$ and α at the end of the stream.*

The input can be thought of as a vector, so we will go back and forth between thinking of the input as a vector and a stream of elements. We know how to solve this problem using $\text{polylog}(n)$ space when $\alpha = \text{poly}(n)$. We now use the decision counters from Section 9.5.1 to solve Problem 9.

Lemma 9.5.4. *There is a randomized algorithm that solves Problem 9 with probability $1 - 1/\text{polylog}(\alpha) \cdot \text{poly}(n)$ and uses $\text{poly}(\log\log\alpha + \log n)$ bits of space.*

We first solve the problem when the support size is 1. The idea is to compute inner products with vectors that reveal the bits of the position of the non-zero element. This idea is called non-adaptive binary search.

Consider the set A of $\log n$ vectors of size n where the i^{th} vector a_i ($i \in \{0, 1, \ldots, \log n - 1\}$) is defined as the vector which has a 1 at an index iff the i^{th} bit of the index is 0. We can also describe it as follows: The vector a_i is made of $n/2^i$ blocks of size 2^i each. Alternately fill the blocks with all 1's and then all 0's. So a_0 will be the vector 10 repeating, a_1 will be the vector 1100 repeating, and $a_{\log n - 1}$ will be the vector with 1's in the first half and 0's in the second half. We use the set A to get the following lemma:

Lemma 9.5.5. *There is a randomized algorithm that solves Problem 9 with the promise that the support size is 1 with probability $1 - 1/\text{polylog}(\alpha) \cdot \text{poly}(n)$ and uses $O(\log n \cdot \log\log\alpha + \log^2 n)$ bits of space.*

Consider the following algorithm for Lemma 9.5.5:

Algorithm 17. A dynamic streaming algorithm for Problem 9 with support size 1.

Input: A vector v specified in a dynamic stream.

Output: The element with non-zero frequency.

During the stream:

1. Store inner products of the frequency vector with all vectors in set A using the decision counters.

Post-Processing:

1. Let s be the $\log n$ bit solution.
2. If the inner product with a_i is 0 then set $s_i = 1$, otherwise set $s_i = 0$.
3. Output s as the non-zero index (where $i = 0$ is the least significant bit).

We first show that this algorithm can be implemented in small space.

Claim 9.5.6. *The space taken by Algorithm 17 is* $O(\log n \cdot \log\log \alpha + \log^2 n)$ *bits.*

Proof. We store $\log n$ decision counters, each taking $O(\log n + \log\log \alpha)$ bits. We do not store the vectors a_i explicitly. When we get element e in the stream, we know which a_i's have a 1 in position e, so we can update those counters. ∎

We now show that we can recover the correct non-zero index with high probability.

Claim 9.5.7. *Algorithm 17 outputs the correct answer with probability* $1 - 1/\text{polylog}\,(\alpha) \cdot \text{poly}(n)$.

Proof. We first condition on none of the decision counters failing. a_i is a vector which has a 1 at some index iff the i^{th} bit of the index is 0. If the inner product of the frequency vector and a_i is 1 then a_i is 1 on the non-zero index. This implies that the i^{th} bit of the non-zero index is 0. Thus, the protocol is deterministically correct after conditioning on none of the decision counters failing. A union bound over the failure probabilities of the counters concludes the proof. ∎

Claim 9.5.6 and Claim 9.5.7 together prove Lemma 9.5.5.

We also need an algorithm that can tell us whether we are in the support 1 case or not. The idea is to partition the coordinates into 4 parts randomly and check the sums of each part. If there are many non-zero coordinates then the hope is that they are spread across multiple partitions, making the sums in multiple partitions non-zero. We use this idea to get the following lemma:

Lemma 9.5.8. *There is a randomized algorithm that can detect whether or not the support size is* 1 *with probability* $1 - 1/\text{polylog}(\alpha) \cdot \text{poly}(n)$ *and uses* $O((\log\log\alpha + \log n)^2)$ *bits of space.*

Consider the following algorithm for Lemma 9.5.8:

Algorithm 18. A dynamic streaming algorithm for detecting a vector with support size 1.

Input: A vector v specified in a dynamic stream.

Output: Whether the support of non-zero elements in v is exactly 1 or not.

Pre-processing:

1. Let $r = O(\log n + \log\log\alpha)$.
2. Create r independent random partitions $V_1^i, V_2^i, V_3^i, V_4^i$ $(i \in [r])$ where every element independently belongs to V_j^i with probability $1/4$.

During the stream:

1. Store inner products z_j^i of the frequency vector with the characteristic vectors of V_j^i using the decision counters for $i \in r$ and $j \in [4]$.

Post-Processing:

1. If there exists i such that more than one z_j^i is non-zero or all z_j^i's are zero then output non-zero support is not 1. Otherwise, output non-zero support is 1.

Claim 9.5.9. *Algorithm 18 can identify whether the support of the non-zero elements is* 1 *or not with probability* $1 - 1/\text{poly}(n) \cdot \text{polylog}(\alpha)$.

Proof. We first condition on the event that none of the counters fail. In the case the support is 1, exactly one z_j^i will be non-zero for all $i \in [r]$. In the case the support is 0, all z_j^i's will be zero for all $i \in [r]$.

Now consider the case where there are $t > 1$ non-zero elements. Let e and e' be two different non-zero elements. Consider the state of the inner products $\tilde{z}_j^i$ without elements e and e'.

- Case 1: If at least 2 counters are zero then with probability at least $1/16$, e and e' are assigned to the counters that are zero, making them non-zero. Thus, there are at least two non-zero counters.
- Case 2: If at least 3 counters are non-zero then with probability at least $1/16$, e and e' are assigned to the same non-zero counter. Thus, there are at least two non-zero counters since two out of the three remain unchanged.

In both cases, we are successful with probability at least $1/16$. Thus, the probability we fail over all iteration is $(1-1/16)^r \leqslant \exp(-r/16) \leqslant 1/\text{poly}(n) \cdot \text{polylog}(\alpha)$. A union bound over the failure probabilities of the counters concludes the proof. ∎

Claim 9.5.10. *Algorithm 18 uses $O((\log n + \log\log \alpha)^2)$ bits of space.*

Proof. There are $4r = O(\log n + \log\log \alpha)$ decision counters each taking $O(\log n + \log\log \alpha)$ bits of space. ∎

Claim 9.5.9 and Claim 9.5.10 together prove Lemma 9.5.8.

We now combine the ideas above to give the final algorithm to solve Problem 9 (for an arbitrary support size). The idea is to reduce to the support size 1 case. Let k be the number of non-zero elements in the frequency vector. If we sample $1/k$ fraction of the coordinates then with constant probability only one of the non-zero elements lies in the support. By repeating this many times we can reduce to the support 1 case with high probability. We do not know k but we can guess it to within a factor of 2 by trying all powers of 2. Also, we can detect when the support size is exactly 1. Consider the final algorithm:

Algorithm 19. A dynamic streaming algorithm for Problem 9.

Input: A vector v specified in a dynamic stream.

Output: An element with non-zero frequency.

During the stream:

For $i = 0$ to $\log n$:

Repeat $r = O(\log n + \log\log \alpha)$ times:

1. Sample every element independently with probability $1/2^i$.
2. Run Algorithm 17 and Algorithm 18 on the sampled subset.

Post-Processing:

1. Find an iteration where Algorithm 18 returns non-zero support 1. If no such iteration exists output "FAIL".
2. Use the corresponding copy of Algorithm 17 to output the non-zero index.

The space used by Algorithm 19 is $\text{poly}(\log\log \alpha + \log n)$.

Claim 9.5.11. *Algorithm 19 uses $O(\log n \cdot (\log\log \alpha + \log n)^3)$ bits of space.*

Proof. We use $r \log n$ copies of Algorithm 17 and Algorithm 18. A copy of Algorithm 17 takes space $O(\log n \cdot \log\log \alpha + \log^2 n)$ bits of space. A copy of Algorithm 18 takes space $O((\log\log \alpha + \log n)^2)$ bits of space. Thus the total space taken is $O(\log n \cdot (\log\log \alpha + \log n)^3)$ bits. ∎

We now show the correctness of Algorithm 19. We start by conditioning on the high probability events of all copies of Algorithm 17 and Algorithm 18. Let k be the size of the non-zero support of the input vector v. Consider the iterations when $i = i^*$ where $k \leqslant 1/p := 2^{i^*} \leqslant 2k$. We need to show that with high probability, in some iteration, exactly one element is sampled.

Claim 9.5.12. *There is an iteration where exactly one element from the non-zero support is sampled with probability $1 - 1/\text{polylog}(\alpha) \cdot \text{poly}(n)$.*

Proof. The claim is trivial when $k = 1$, so we consider $k \geqslant 2$. The probability that exactly one element is sampled is:

$$\Pr(\text{exactly one element is sampled}) = k \cdot p \cdot (1-p)^{k-1}$$

$$\geqslant k \cdot \frac{1}{2k} \cdot (1-1/k)^k$$
$$\geqslant 1/16. \qquad (k \geqslant 2 \text{ implies } (1-1/k)^k \geqslant 1/8)$$

We fail when this does not happen in any iteration. The probability of failure is:

$$\Pr(\text{failure}) \leqslant (1-1/16)^r \leqslant 1/\text{polylog}\,(\alpha) \cdot \text{poly}(n).$$

Thus, there is an iteration where exactly one element from the non-zero support is sampled with probability $1 - 1/\text{polylog}\,(\alpha) \cdot \text{poly}(n)$. ∎

This means that with high probability, there is an iteration where exactly one element from the non-zero support is sampled. For this iteration with high probability, the corresponding copy of Algorithm 18 will output non-zero support 1, and the corresponding copy of Algorithm 17 will output the non-zero index. Also, for all other iterations where the support size is not 1, the corresponding copy of Algorithm 18 will output non-zero support is not 1 with high probability. A union bound over the failure probabilities of all copies of Algorithm 18 and Algorithm 17 proves the correctness of Algorithm 19 in Lemma 9.5.4. Claim 9.5.11 proves the space bound of Algorithm 19 in Lemma 9.5.4. Therefore, we have proved Lemma 9.5.4.

Note that the number of random bits used is more than the space bound. But we can fix this by using Nisan's pseudorandom generator (Fact 9.3.5). Using this increases the space by a factor of $O(\log n)$ and makes the true randomness needed fit in the space bound.

9.5.2 Edge Connectivity

A Certificate of Edge Connectivity We present a certificate of k-edge-connectivity in this section. Our algorithm and analysis are very similar to the k-vertex connectivity algorithm of [AS23a].

Algorithm 20. An algorithm for computing a certificate of k-edge-connectivity.

Input: A graph $G = (V, E)$ and an integer k.

Output: A certificate H for k-edge-connectivity of G.

1. For $i = 1, 2, \ldots, r := (200k \ln n)$ do the following:
    - Let E_i be a subset of E where each edge is sampled independently with

probability $1/k$.

- Let $G_i = (V, E_i)$ be the subgraph of G containing edges of E_i.
- Compute a spanning forest T_i of G_i.

2. Output $H := T_1 \cup T_2 \cup \ldots \cup T_r$ as a certificate for k-edge-connectivity of G.

The algorithm of [AS23a] samples vertices with probability $1/k$ and stores spanning forests over the sampled induced subgraph. The number of iteration in their algorithm is also larger by a factor of k because in each iteration they are working with a smaller graph on roughly n/k vertices.

The following theorem proves the main guarantee of this algorithm.

Theorem 27. *Given any graph $G = (V, E)$ and any integer $k \geqslant 1$, Algorithm 20 outputs a certificate H of k-edge-connectivity of G with $O(kn \cdot \log n)$ edges with high probability.*

The analysis in the proof of Theorem 27 is twofold. We first show that pairs of vertices that are at least $2k$-edge-connected in G stay k-edge-connected in H. Secondly, we show that edges whose endpoints are not $2k$-edge-connected in G will be preserved in H. Putting these together, we then show that H is a certificate for k-edge-connectivity of G and has at most $\widetilde{O}(kn)$ edges.

We start by bounding the number of edges of the certificate H.

Lemma 9.5.13. *The certificate H in Algorithm 20 has $O(kn \cdot \log n)$ edges.*

Proof. Each spanning forest T_i has at most $n - 1$ edges. Thus, the total number of edges in H is at most $200kn \ln n$. ∎

We now prove the correctness of this algorithm in the following lemma.

Lemma 9.5.14. *Subgraph H of Algorithm 20 is a certificate of k-edge-connectivity for G with high probability.*

Lemma 9.5.14 will be proven in two steps. We first show that every pair of vertices that have at least $2k$ edge-disjoint paths between them in G have at least k edge-disjoint paths in H with high probability.

Lemma 9.5.15. *Every pair of vertices s, t in G that have at least $2k$ edge-disjoint paths between them in G have at least k edge-disjoint paths in H with high probability.*

We then show that every edge whose endpoints have less than $2k$ edge-disjoint paths between them in G will belong to H as well.

Lemma 9.5.16. *Every edge $(s, t) \in G$ that has less than $2k$ edge-disjoint paths between its endpoints in G belongs to H also with high probability.*

The proofs of these lemmas appear in the next two subsections. We first use these lemmas to prove Lemma 9.5.14 and conclude the proof of Theorem 27.

Proof of Lemma 9.5.14. We first condition on the events in Lemma 9.5.15 and Lemma 9.5.16 both of which happen with high probability. We need to show that H is k-edge-connected iff G is k-edge-connected. If H is k-edge-connected then G is also k-edge-connected simply because H is a subgraph of G.

We now assume towards a contradiction that G is k-edge-connected, but H is not. This means that there is a cut S of size at most $k-1$, i.e. removing the set of edges X going out of S disconnects H. Since G is k-edge-connected, the cut S has at least k edges going out of it so deleting the set of edges X cannot disconnect G and thus G has an edge $e = (s, t)$ between S and $T := V - S$ (see Figure 9.3).

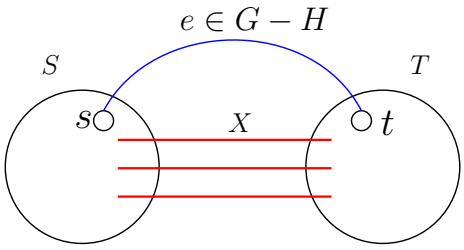


Figure 9.3: An illustration of the cut S in G and H. After X is deleted, there are no edges between S and T in H, while G has at least one edge $e = (s, t)$ between S and T, to ensure its k-edge-connectivity as $|X| < k$.

We now consider two cases.

- Case 1: s and t have at least $2k$ edge-disjoint paths between them in G.
  We have conditioned on the event in Lemma 9.5.15, so we can say that s and t have at least k edge-disjoint paths in H. Deleting X can destroy at most $|X| \leqslant k-1$ of these paths in H. This implies that there is still an s-t path in $H - X$ and thus there is an edge between S and T in $H - X$, a contradiction.

- Case 2: s and t have less than $2k$ edge-disjoint paths between them in G.
  Since there are fewer than $2k$ edge-disjoint paths between s and t in G, by conditioning on the event of Lemma 9.5.16, e would be preserved in H, a contradiction with H having no edge between S and T.

In conclusion, we get that H is a certificate of k-edge-connectivity for G with high probability. ∎

Theorem 27 now follows immediately from Lemma 9.5.13 and Lemma 9.5.14.

Before moving on from this section, we present the following corollary of Algorithm 20 that allows for using this algorithm for some other related problems in dynamic streams as well.

Corollary 9.5.17. *The subgraph H output by Algorithm 20 with high probability satisfies the following guarantees:*

(*i*) *For any pair of vertices s, t in G, there are at least k edge-disjoint s-t paths in G iff there at least k edge-disjoint s-t paths in H (this holds even if G is not k-edge-connected).*

(*ii*) *Every cut of H with size less than k is a cut in G with size less than k and vice versa (this means all cuts of G are preserved in H and no new ones are created as long as their size is less than k).*

The proof of this corollary is identical to that of Lemma 9.5.14 and is thus omitted.

Proof of Lemma 9.5.15. We prove Lemma 9.5.15 in this part following the same approach as in [AS23a]. For this proof, without loss of generality, we can assume that $k > 1$: for $k = 1$, each graph G_i is the same as G and thus the spanning forest computes an s-t path which will be added to H, trivially implying the proof.

Fix any pair of vertices s, t with at least $2k$ edge-disjoint paths between them. We choose an arbitrary set X of $k-1$ edges, and the goal is to show that s and t remain connected in the graph $H - X$ with very high probability. We do so by showing that out of the at least k edge-disjoint paths between s and t in $G - X$, with probability $1 - n^{-\Theta(k)}$, at least one of them is entirely sampled as part of the *subset* of G_i's for $i \in [r]$ that do not contain any edges from X. This will be sufficient to prove the existence of an s-t path in $H - X$. A union bound over the $\binom{n^2}{k-1}$ choices of X and $\binom{n}{2}$ pairs s, t concludes the proof.

Fix X as a set of $k-1$ edges. Define:

$$I(X) := \{i \in [r] : E_i \cap X = \emptyset\}; \tag{9.3}$$

that is, the indices of sampled graphs in $G_1, \ldots, G_r$ that contain no edges from X. We first argue that $|I(X)|$ is large with very high probability.

Claim 9.5.18. $\Pr(|I(X)| \leqslant r/8) \leqslant n^{-5k}$.

Proof. Fix any index $i \in [r]$ and an edge $e \in X$. The probability that e is not sampled in E_i is $(1-1/k)$ by definition and thus,

$$\Pr(E_i \cap X = \emptyset) = (1-1/k)^{k-1} \geqslant 1/4,$$

given that $k > 1$ (as argued earlier) and the sampling of edges being independent in E_i. Therefore, we have,

$$\mathbb{E}\,|I(X)| = r \cdot (1-1/k)^{k-1} \geqslant r/4.$$

By an application of the Chernoff bound (Proposition 2.4.3) with $\mu_L = r/4$ and $\varepsilon = 1/2$, we have,

$$\Pr(|I(X)| \leqslant r/8) \leqslant \exp(-r/4 \cdot 1/10) < n^{-5k}. \qquad \blacksquare$$

In the rest of the proof we condition on the event that $|I(X)| \geqslant r/8$. To continue, we need some definitions. There are more than k edge-disjoint paths between s and t in $G - X$ since there were $2k$ of them in G and only $k-1$ edges (set X) are deleted. Choose k of them arbitrarily denoted by $P_1(X), \ldots, P_k(X)$ (see Figure 9.4 for an illustration).

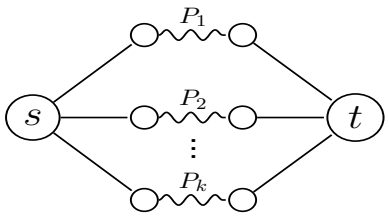


Figure 9.4: An illustration of the k edge-disjoint s-t paths $P_1(X), P_2(X), \ldots, P_k(X)$. Note that the paths can share vertices but are edge-disjoint.

We define the notion of "preserving" a path.

Definition 9.5.19. *Let $G_X := \cup_{i \in I(X)} G_i$ be the union of graphs indexed in Eq (9.3). We say that a path P in $G - X$ is* **preserved** *in G_X iff for every edge $e \in P$, there exists at least one $i \in I(X)$ such that $e \in G_i$; in other words, the entire path P belongs to G_X.*

We are going to show that with very high probability, at least one path $P_j(X)$ for $j \in [k]$ is preserved by G_X. Before that, we have the following claim that allows us to use this property to conclude the proof.

Claim 9.5.20. *If any s-t path $P_j(X)$ for $j \in [k]$ is preserved in G_X then s and t are connected in $H - X$.*

Proof. Given that $P := P_j(X)$ is preserved, we have that for any edge $e = (u, v) \in P$, there is some graph G_i for $i \in I(X)$ that contains e. This means that u, v are connected in G_i which in turn implies that the spanning forest T_i of G_i contains a path between u and v. Moreover, since $i \in I(X)$, we know that G_i and hence T_i contain no edges of X and thus u and v are connected in $T_i - X$ as well. Stitching together these u-v paths for every edge $(u, v) \in P$ then gives us a walk between s and t in $H - X$, implying that s and t are connected in $H - X$. ∎

We will now prove that some path $P_j(X)$ for $j \in [k]$ is preserved with very high probability.

Claim 9.5.21. *Conditioned on $|I(X)| \geqslant r/8$, we have the following:*

$$\Pr(P_j \not\subseteq G_X \text{ for at least } k/2 \text{ values of } j \in [k]) \leqslant n^{-11k}.$$

Proof. To start the proof, note that even conditioned on a choice of $I(X)$, the edges in each path $P_j(X)$ appear independently in each graph G_i for $i \in I(X)$. This is because these paths do not intersect with X and by the independence in sampling of each graph G_i for $i \in [r]$. Moreover, given that these paths are edge-disjoint, the choices of their edges across each graph G_i for $i \in [r]$, are independent. We crucially use these properties in this proof.

Each edge in P_j is not present in G_i with probability $(1 - 1/k)$ and hence is not present in G_X with probability $(1 - 1/k)^{|I(X)|}$. Hence, by the union bound,

$$\Pr(P_j \not\subseteq G_X) \leqslant |P_j| \cdot (1 - 1/k)^{|I(X)|} \leqslant n \cdot (1 - 1/k)^{r/8} \leqslant n \cdot \exp(-200k \ln n/8k) = n^{-24}.$$

Finally, note that since the paths P_j for $j \in [k]$ are edge-disjoint, the probability of the above event is independent for each one. Thus,

$$\Pr(P_j \not\subseteq G_X \text{ for at least } k/2 \text{ values of } j \in [k]) \leqslant \binom{k}{k/2} \left(n^{-24}\right)^{k/2} \leqslant 2^k \cdot n^{-12k} \leqslant n^{-11k}.$$

As such, the entire path P_j will lie inside G_X for at least $k/2$ values of $j \in [k]$ with very high probability. ∎

By union bound over the events of Claims 9.5.18 and 9.5.21, we have that there exists an index $j \in [k]$ such that the path $P_j(X)$ is preserved in G_X. Thus, by Claim 9.5.20, for a fixed choice of X, and s, t, the probability that s and t are not connected in $H - X$ is at most n^{-11k}. A union bound over the n^{2k} choices of X and n^2 choices of s, t, then implies that the probability that even one such choice of X and s, t exists is at most n^{-7k}. This completes the proof of Lemma 9.5.15. ∎

Proof of Lemma 9.5.16. We now prove Lemma 9.5.16. For this proof also, without loss of generality, we can assume that $k > 1$: for $k = 1$, each graph G_i is the same as G and thus the spanning forest uses the only s-t path, namely, the edge (s, t) (as s and t can only be 1-connected through the edge (s, t)) which will be added to H, thus trivially implying the proof. We now consider the main case.

Fix any pair of vertices $s, t \in G$ which have less than $2k$ edge-disjoint paths between them. We know that deleting some set X of at most $2k - 1$ edges should disconnect s and t. For any $i \in [r]$, we call the graph G_i **good** if it samples the edge (s, t) and does not sample any edge from X. See Figure 9.5 for an illustration.

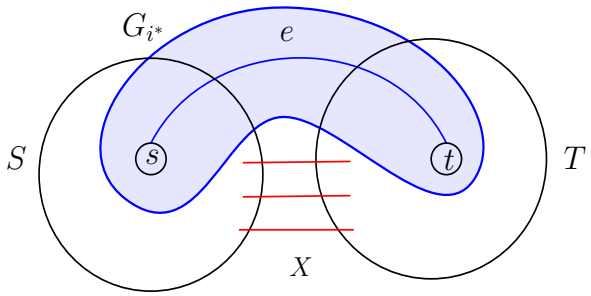


Figure 9.5: An illustration of a good graph G_{i^*} wherein the edge $e = (s, t)$ is sampled and all the edges in set X are not. Thus, none of the s-t paths, except for the edge e, exist in G_{i^*}. Therefore, the spanning forest T_{i^*} necessarily contains the edge $e = (s, t)$.

We have,

$$\Pr(G_i \text{ is good}) \geqslant 1/k \cdot (1 - 1/k)^{2k-1} \geqslant 1/8k. \qquad (\text{as } k \geqslant 2 \text{ so } (1 - 1/k)^{2k-1} \geqslant (1/2)^3)$$

Given the independence of choices of G_i for $i \in [r]$, we have,

$$\Pr(\text{No } G_i \text{ is good}) \leqslant (1 - 1/8k)^r \leqslant \exp(-200k \ln n/8k) = n^{-25}.$$

Therefore, there is a graph G_{i^*} for $i^* \in [r]$ where (s, t) is sampled but X is not (see Figure 9.5). This means that the spanning forest T_{i^*} has to contain the edge (s, t) as there

is no other path between s and t (we have effectively "deleted" X by not sampling it). Thus, the edge (s,t) belongs to H with probability at least $1-n^{-25}$. A union bound over all possible pairs $s,t \in G$ concludes the proof. ∎

The SGT Streaming Algorithm We present our single pass SGT streaming algorithm for k-edge-connectivity in this section. The parameter for SGT streaming is α i.e. the frequencies are bounded between $-\alpha$ and α. We note that the $\widetilde{O}(\cdot)$ notation will hide $\log n$ and $\log\log\alpha$ factors. The algorithm outputs a certificate of k-edge-connectivity for the input graph at the end of the stream. Thus, by the definition of the certificate, to know whether or not the input graph is k-edge-connected, it suffices to test if the certificate is k-edge-connected, which can be done at the end of the stream using any offline algorithm. Consider the following theorem:

Theorem 28. *There is a randomized SGT streaming algorithm that given an integer $k \geqslant 1$ before the stream and a graph $G=(V,E)$ in the stream, outputs a certificate H of k-edge-connectivity of G with high probability using $\widetilde{O}(kn)$ bits of space.*

This algorithm is just an implementation of Algorithm 20 in SGT streams. We fix the edge sets E_i in Algorithm 20 before the stream so the only thing we need to specify is how we compute the spanning forests during the stream. We compute a spanning forest T_i of G_i in the stream using the dynamic streaming algorithm in Fact 9.3.10 with strong ℓ_0-samplers (Section 9.5.1). After the stream, we output the certificate H. This completes the description of the streaming algorithm.

We start by bounding the space of this algorithm.

Lemma 9.5.22. *This algorithm uses $\widetilde{O}(kn)$ bits of space.*

Proof. During the stream, we run a spanning forest algorithm for each graph G_i for $i \in [r]$. The algorithm of Fact 9.3.10 with strong ℓ_0-samplers takes at most $\widetilde{O}(n)$ bits of space. So the total space taken is $\widetilde{O}(r \cdot n) = \widetilde{O}(kn)$ bits. ∎

We are now ready to prove Theorem 28.

Proof of Theorem 28. By Lemma 9.5.22, this algorithm uses $\widetilde{O}(kn)$ bits of space. The algorithm is also correct with high probability by Theorem 27. Moreover, none of the spanning forest algorithms of Fact 9.3.10 fail with high probability (by a union bound over the failure probabilities of the r spanning forest algorithms).

The strong ℓ_0-samplers work in the SGT model because they can handle large values and they do not distinguish between different non-zero values (even negative values). Also, adding the samplers for the neighborhood of vertices in a component gives a sampler for the edges going out of the component because the internal edges cancel out (similar to traditional ℓ_0-samplers). Finally, we do not subtract edges from samplers to delete edges because to delete an edge, we need to know the exact frequency of the edge (this was the problem with the original dynamic streaming algorithm for edge connectivity [AGM12a]). Thus, the streaming algorithm uses $\widetilde{O}(kn)$ bits of space and, by union bound, with high probability outputs a certificate of k-edge-connectivity. ∎

Note that the number of random bits used is more than the space bound. But we can fix this by using Nisan's pseudorandom generator (Fact 9.3.5). Using this increases the space by a factor of $O(\log n)$ and makes the true randomness needed fit in the space bound.

We also note that using the strong ℓ_0-samplers we get a $\widetilde{O}(n)$ space algorithm for connectivity and a $\widetilde{O}(kn)$ space algorithm for k-vertex-connectivity in the SGT streaming model.

9.6 Annotated Streaming Schemes for Dynamic Graph Streams

9.6.1 The Layering Lemma

In this subsection, we prove some simple properties of k-connected graphs and prove important lemmas called layering lemmas for vertex and edge connectivity.

Properties of k-connected graphs

Claim 9.6.1. *Let G be k-vertex-connected. Consider a new vertex v and attach it to $\geqslant k$ arbitrary vertices of G and call this new graph G'. Then G' is k-vertex-connected.*

Proof. Consider a vertex cut X of size at most $k-1$. X cannot disconnect any pair of vertices in G since G is k-vertex-connected. If v is isolated on one side of the cut then it must have a neighbor on the other side since X has at most $k-1$ vertices and v has $\geqslant k$ neighbors. ∎

Claim 9.6.2. *Let $G = (V, E)$ be a graph and let $T \subseteq V$ be a set of vertices. We know that there are at least k vertex-disjoint (edge-disjoint) paths from every vertex of T to a special*

vertex $t \notin T$. If a vertex $v \notin T \cup \{t\}$ has k vertex-disjoint (edge-disjoint) paths to T then there are k vertex-disjoint (edge-disjoint) paths between v and t.

Proof. Consider any vertex cut X of size at most $k-1$. We will show that v and t are connected after deleting X. X has at most $k-1$ vertices so there is a path P from v to T that has no vertices of X. Let $t' \in T$ be the vertex path P ends on. We know that there are k vertex-disjoint paths between t' and t (by assumption) thus after deleting X there is at least one path that has no vertices of X. This means that after deleting X there is a path from v to t via t' implying that there are k vertex-disjoint paths between v and t. ∎

Claim 9.6.3. *Let $G = (V, E)$ be a graph and let X be a fixed minimum vertex cut of G. If there is a vertex $t \in V - X$, such that there are k vertex-disjoint paths from t to v for all $v \in V - \{t\}$ then G is k-vertex-connected (in other words $|X| \geqslant k$).*

Proof. Assume towards a contradiction that the vertex cut X has size at most $k-1$. Let T be a connected component containing t after X is deleted and let $S := V - X - T$. Consider an arbitrary vertex s in S. We know that there are k vertex disjoint paths between s and t. So deleting $k-1$ vertices cannot disconnect v and t_i, giving us a contradiction. ∎

Corollary 9.6.4. *Let $G = (V, E)$ be a graph and let $t_1, t_2, \ldots, t_k \in V$ be fixed vertices. If for all i, G has k vertex-disjoint paths from t_i to v for all $v \in V - \{t_i\}$ then G is k-vertex-connected.*

Proof. Assume towards a contradiction that there is a vertex cut X of size at most $k-1$ that when deleted disconnects the graph. There is a t_i that is not in X since there are k special vertices and $|X| \leqslant k-1$. Applying Claim 9.6.3 with X and t_i implies that $|X| \geqslant k$, giving us a contradiction. Thus, G is k-vertex-connected. ∎

The Layering Lemma

In this subsection, we prove our layering lemmas for vertex and edge connectivity that will be useful in our algorithms. We will show complete proofs for vertex-connectivity and then only mention the differences for edge-connectivity because the proofs are very similar. Consider the following lemmas that show short proofs for k disjoint paths from a fixed vertex to all other vertices.

Lemma 9.6.5. *Let $G = (V, E)$ be a k-vertex-connected graph, and let $t \in V$ be an arbitrary vertex. There is a $\widetilde{O}(kn)$ size proof which shows that there are k vertex-disjoint paths from t to v for all $v \in V - \{t\}$.*

Lemma 9.6.6. *Let $G = (V, E)$ be a k-edge-connected graph, and let $t \in V$ be an arbitrary vertex. There is a $\widetilde{O}(k^2 n)$ size proof which shows that there are k edge-disjoint paths from t to v for all $v \in V - \{t\}$.*

We will show that a short proof exists using the probabilistic method. We first set up the structure of the proof. Note that we will write disjoint paths without specifying edge-disjoint or vertex-disjoint to mean either of those because the layering structure for both is the same. We want a proof that shows that the special vertex t has k disjoint paths to all vertices in $V - \{t\}$. The idea is to first build a set of vertices T_0 and show that every vertex in T_0 has k disjoint paths to t. The next step is to inductively build sets $T_1, T_2, \ldots T_\ell$ and show that every vertex in T_i has k disjoint paths to the set T_{i-1}. Using Claim 9.6.2, this shows that every vertex in T_i has k disjoint paths to t (since every vertex in T_{i-1} has k disjoint paths to t). We keep doing this till we cover all the vertices using the sets T_i and thus, everyone has k disjoint paths to t. This proves the correctness of Lemma 9.6.5 and Lemma 9.6.6.

We now show how these sets are constructed and bound the proof size. Let $\ell := \log(n/k)$. For $i \in [0, \ell]$, let T_i^L, T_i^R be the sets where every vertex is independently sampled with probability $p_i := \frac{k}{n} \cdot 2^i$. $T_i = T_i^L \cup T_i^R$. Note that we sample all vertices in T_ℓ, so we cover all the vertices. Also, note that it is okay if a vertex is sampled in multiple T_i's. The expected size of T_i is $2^{i+1}k$ for all $i \in [0, \ell]$. We will now show that for any vertex, the proof for k disjoint paths to T_i is small in expectation.

Claim 9.6.7. *For any vertex v the proof for k disjoint paths to T_i takes size at most $\frac{n}{2^i}$ words in expectation.*

Proof. Consider the vertex v and its k disjoint paths $P_1, P_2, \ldots P_k$ to T_i^L (this exists because the graph is k-connected). For any path P_j, we truncate it at the first occurrence of a vertex from T_i^R (since we want paths to $T_i = T_i^L \cup T_i^R$). We now use the randomness of T_i^R. Every vertex on P_j is independently sampled in T_i^R with probability $p_i = \frac{k}{n} \cdot 2^i$. Thus, in expectation, the truncated length of P_j is at most $\frac{1}{p_i} = \frac{n}{k \cdot 2^i}$. Therefore, in expectation, the k disjoint paths to T_i together have size at most $\frac{n}{2^i}$ (by linearity of expectation). ∎

This means that in expectation, the total proof size for k-vertex-connectivity is $O(kn \log(n/k))$ words.

Claim 9.6.8. *The expected total proof size for k-vertex-connectivity is $O(kn \log(n/k))$ words.*

Proof. The proof of k vertex-disjoint paths for a fixed vertex in T_0 takes size at most $O(n)$ words because the summed length of all paths is $O(n)$ (any vertex belongs to at most one vertex-disjoint path). This gives a total expected size of $O(kn)$ words for all vertices in T_0. The proof for vertices in T_{i+1} takes size at most $\frac{n}{2^i}$ (by Claim 9.6.7) which gives a total expected size of $2^{i+2}k \cdot \frac{n}{2^i} = 4kn$ words for all vertices in T_{i+1}. We have $\ell = \log(n/k)$ sets T_i implying the claim. ∎

Claim 9.6.9. *The expected total proof size for k-edge-connectivity is $O(k^2 n \log(n/k))$ words.*

Proof. The only difference from the proof of Claim 9.6.8 is that the proof for vertices in T_0 takes size at most $O(kn)$ since each path can be of size at most n. This happens because edge-disjoint paths can use the same vertices. This gives us an overall proof size of $O(k^2 n \log(n/k))$ words. ∎

If we can show that vertices T_0 have k edge-disjoint paths to t in smaller space then we can improve the overall proof size (since this is the main bottleneck). For instance, if the graph has a subgraph on $O(k)$ vertices that is k-edge-connected then that set could be T_0 and the overall proof size would be $O(kn \log(n/k))$ (this would increase the verification space by an additive $O(k^2)$ words).

Proof of Lemmas 9.6.5 and 9.6.6. We have proved the correctness and the size bounds in expectation. There must be some random string that achieves a proof size of at most the expected size (by definition) implying that there exists a proof of size $O(kn \log(n/k))$ words for k-vertex-connectivity and a proof of size $O(k^2 n \log(n/k))$ words for k-edge-connectivity (which an all powerful prover can find). Thus, the prover sends this proof to the verifier. ∎

We now show that the prover can send some auxiliary information so that the verifier can verify the proof.

Claim 9.6.10. *The layering proof of Lemma 9.6.5 for k-vertex-connectivity can be verified using a $[kn + h, v]$-scheme for any h, v such that $h \cdot v = n^2$.*

Proof. The prover sends k vertex-disjoint paths for each vertex using the layering idea mentioned above (Lemma 9.6.5). There are three things that need to be verified. First, all the edges that the prover sends should be a part of the input graph. Second, the edges sent by the prover should form paths. Finally, the paths for a vertex should be vertex disjoint.

The prover sends a list of all edges used in the proof along with their multiplicities in $\widetilde{O}(kn)$ space. It is easy to do a subset check for the edges (ignoring multiplicities) using an $[h, v]$-scheme since there are at most n^2 edges (Fact 9.3.7). The problem now is to check if the prover was truthful about the multiplicities, which can be done using the ℓ_0-sampler trick using $\widetilde{O}(1)$ space (Proposition 9.3.11). The verifier can maintain an ℓ_0-sampler where he inserts all edges along with their multiplicities when they are provided upfront. He then deletes all edges used in the proof of k vertex-disjoint paths for each vertex. Finally, he checks whether the sampler is empty. If it is not then the two sets of edges are not the same and thus, the verifier catches the lying prover. The probability of failure is at most $1 - 1/\text{poly}(n)$.

The prover will send the edges of the paths one path at a time, so it is easy to verify that the edges form a path. Finally, we need to verify that the paths for every vertex are vertex disjoint. For this, the prover sends the vertices he is going to use in the proof upfront in increasing order. The verifier can verify this in $\widetilde{O}(1)$ space by just checking the increasing order. The problem now is to check if the prover lied so we can run an equality check on the stream of vertices sent upfront and the vertices used in the proof. This can be done using the ℓ_0-sampler trick in $\widetilde{O}(1)$ verification space (Proposition 9.3.11). We have to do this for all vertices so the worry is that the proof size or verification space might blow up. Since the auxiliary information is just repeating the vertices used in the proof in sorted order, adding this auxiliary information can at most double the space used. Also, the space on the verifier side is $\widetilde{O}(1)$ words for each vertex, but this space can be reused implying $\widetilde{O}(1)$ words of verification space for the entire disjointness check.

Therefore, the proof along with the auxiliary information takes size $\widetilde{O}(kn) + \widetilde{O}(h)$ bits and can be verified in $\widetilde{O}(v) + \widetilde{O}(1)$ bits of space, proving the $[kn + h, v]$-scheme. ∎

Claim 9.6.11. *The layering proof of Lemma 9.6.6 for edge-connectivity can be verified using a $[k^2n + h, v]$-scheme for any h, v such that $h \cdot v = n^2$.*

The proof is almost identical with the only difference being that the proof for k-edge-connectivity is of size $\widetilde{O}(k^2n)$ opposed to $\widetilde{O}(kn)$ for k-vertex-connectivity. This happens because if we consider k-vertex disjoint paths then their total length has to be $O(n)$ because every vertex can be used at most once. However, if we have k edge disjoint paths then the same vertices can be used in multiple paths making each of the k paths potentially of length $O(n)$. We can also construct k edge disjoint paths between a pair of vertices where the length of each path is $\Omega(n)$.

Consider the following construction on at most $2n$ vertices where n is prime and let $k < n/2$. The vertex set consists of vertices in $U := \{u_0, u_2, \ldots, u_{n-1}\}, V := \{v_1, v_2, \ldots, v_k\}$,

and $T := \{t\}$. We will show k edge-disjoint paths between u_0 and t where each path has length $\Omega(n)$. Path i starts from u_0 and goes to u_i then u_{2i} and so on for $n-1$ steps. Note that these coordinates in the subscript are mod n. After the $n-1$ steps, there is an edge to v_i and finally to t. This completes the description of all the paths.

There walks clearly have length $\Omega(n)$ so we first show that these walks are also paths. These just follows from n being prime. All $j \cdot i \pmod{n}$ are different for $j \in [n-1]$. We now show why the paths are edge disjoint. This is because an edge between j_1 and j_2 can only be in path $i = j_1 - j_2$ where $j_1 > j_2$. Thus, the same edge cannot belong to two different paths. This concludes the proof of the hard construction.

We now consider the following corollary of Claim 9.6.10.

Corollary 9.6.12. *Let $G = (V, E)$ be a k-vertex-connected graph and let $t_1, t_2, \ldots, t_r \in V$ be arbitrary distinct vertices. There is a $[r \cdot (kn + h), v]$-scheme for all h, v such that $h \cdot v = n^2$, which shows that for all $i \in [r]$ there are k vertex-disjoint paths from t_i to v for all $v \in V - \{t_i\}$.*

Proof. If the prover repeats Claim 9.6.10 r times and the verifier reuses his space then the proof size blows up by a factor of r but we only get an additional r words in the verification space. Once the proof for a vertex has been verified, the memory can be emptied and reused because the verifier only has to remember that the proof has been verified for that particular vertex. So the verifier just remembers the vertices $t_1, t_2, \ldots, t_r$ and checks if they are distinct. But the distinctness check can be done more efficiently.

The verifier keeps a counter which he increments after he has verified the proof for a vertex. After the entire proof is sent the verifier checks if the counter reached r. The only thing left to do is check the distinctness. In parallel, the verifier runs an $[h, v]$-scheme ($h \cdot v \geqslant n$) for duplicate check (Fact 9.3.8). Thus the additional space usage is only $\widetilde{O}(1) + \widetilde{O}(v)$ bits. ∎

We now show that Lemma 9.6.5 and Claim 9.6.10 work even when t is a set of vertices (of size at least k) instead of a single vertex.

Corollary 9.6.13. *Let $G = (V, E)$ be a k-vertex-connected graph and let $T \subseteq V$ be an arbitrary set of vertices with $|T| \geqslant k$. There is a $[kn + h, v]$-scheme for any h, v such that $h \cdot v = n^2$, which shows that there are k vertex-disjoint paths from the set T to v for all $v \in V - T$.*

Proof. Create a new graph G' by introducing a new vertex t and connecting it to all vertices in T. We know by Claim 9.6.1 that G' is k-vertex-connected. We can now apply

Lemma 9.6.5 and Claim 9.6.10 for vertex t and this gives us a proof of k vertex-disjoint paths from t to all vertices in $V - \{t\}$.

Let v be a vertex in $V - T$. v has k vertex-disjoint paths to t. The last edges of all these paths are from $N(t) = T$ to t implying that v has k vertex-disjoint paths to the set T. ∎

We get a similar corollary for k-edge-connectivity.

Corollary 9.6.14. *Let $G = (V, E)$ be a k-edge-connected graph and let $T \subseteq V$ be an arbitrary set of vertices with $|T| \geqslant k$. There is a $[k^2n + h, v]$-scheme for any h, v such that $h \cdot v = n^2$, which shows that there are k edge-disjoint paths from the set T to v for all $v \in V - T$.*

9.6.2 Vertex Connectivity Schemes

In this section, we discuss schemes for k-vertex-connectivity.

Theorem 22. *Under dynamic graph streams on n-node graphs, there exists a $[k\cdot(h+kn), v]$-scheme for k-vertex-connectivity for any h, v such that $h \cdot v = n^2$. In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(k^2n)$.*

We start by showing a very simple $[n^2k, n]$-scheme.

Simple Algorithm

Assume that the graph $G = (V, E)$ is k-vertex-connected. The simple algorithm is to fix k special vertices $t_1, t_2, \ldots t_k$ and show that all other vertices have k vertex-disjoint paths to each t_i for $i \in [k]$. This is enough to show that the graph is k-vertex-connected (by Corollary 9.6.4)

We know that showing k vertex-disjoint paths between two vertices takes $O(n)$ proof size (any vertex can belong to at most one vertex-disjoint path). We can check if the edges used in the proof belong to the graph using an $[n, n]$-scheme (Fact 9.3.7). Thus showing that all vertices have k vertex-disjoint paths to each t_i has a $[kn^2, n]$-scheme.

If the graph is not k-vertex-connected then the prover can send the vertex cut X along with a connected component S that is formed after deleting X in $\widetilde{O}(n)$ space. The verifier then has to check whether there are any edges between S and $T := V - X - S$. So the only allowed edges are within S, within T or incident on X. The verifier constructs a graph G' in a streaming fashion with all such edges and then checks if the edges in input graph are

a subset of G'. This can be done using an $[h, v]$-scheme for subset check for any h, v such that $h \cdot v = n^2$ (Fact 9.3.7). We use a $[kn, n/k]$-scheme to conclude the simple scheme.

We now improve this algorithm using the layering lemma (Lemma 9.6.5) and give a $[k \cdot (h + kn), v]$-scheme.

Layering Algorithm

If the graph $G = (V, E)$ is not k-vertex-connected, then we can use the proof of the simple algorithm which has an $[n + h, v]$-scheme. In the other case, the issue with the simple algorithm was that the proofs for all the vertices were of size $O(n)$. The goal would be to reduce the proof sizes for some vertices.

Proof of Theorem 22. Fix k arbitrary vertices $t_1, t_2, \ldots, t_k$. Use Corollary 9.6.12 to get a proof of k vertex-disjoint paths from t_i to every vertex in $V - \{t_i\}$ for all $i \in [k]$. Thus, the proof size for all t_i together is $\widetilde{O}(k \cdot (h + kn))$ words and can be verified using $\widetilde{O}(v)$ space for any h, v such that $h \cdot v = n^2$. This is a valid proof by Corollary 9.6.4 which shows that G is k-vertex-connected. Thus, we get a $[k \cdot (h + kn), v]$-scheme for k-vertex-connectivity for any h, v such that $h \cdot v = n^2$, proving Theorem 22. ∎

Notice that the product of the proof size and the verification space is kn^2 (when $h \geqslant kn$) which seems suboptimal. There are two relaxations that can drop this extra factor of k. We show them in the next two subsections.

Distinguishing between k and $2k$ vertex-connectivity

In this subsection, we give a $[kn + h, v]$-scheme for any h, v such that $h \cdot v = n^2$, for distinguishing between vertex-connectivity $< k$ and $\geqslant 2k$. This is the promised-gap k-vertex-connectivity problem with $\varepsilon = 1$ [AS23a, GMT15].

Lemma 9.6.15. *Under dynamic graph streams on n-node graphs, there exists an $[kn+h, v]$-scheme for distinguishing between vertex-connectivity $< k$ and $\geqslant 2k$, for any h, v such that $h \cdot v = n^2$.*

Proof. If the graph $G = (V, E)$ is not k-vertex-connected we can use the $[n + h, v]$-scheme of the simple algorithm where $h \cdot v = n^2$.

If the graph $G = (V, E)$ is $2k$-vertex-connected then we fix two disjoint sets T_L and T_R containing $2k$ arbitrary vertices each. We use Corollary 9.6.13 to show $2k$ vertex-disjoint paths from every vertex in $V - T_L$ to the set T_L. This can be done using a $[kn+h, v]$-scheme

for any h, v such that $h \cdot v = n^2$. We do the same for T_R. This is enough to show that G is at least k-vertex-connected (Claim 9.6.16) implying that G is $2k$-vertex-connected (promise). ∎

Claim 9.6.16. *The proof shows that G is at least k-vertex-connected.*

Proof. Assume towards a contradiction that there is a vertex cut X of size $< k$. We will show that all the vertices are connected after deleting X implying that G is at least k-vertex-connected.

Consider two arbitrary vertices $u, v \in T_L$. The proof for T_R shows that each of u, v have $2k$ vertex-disjoint paths to T_R. X has at most $k-1$ vertices so each of u, v reach at least $k+1$ vertices in T_R after deleting X. By the pigeonhole principle there is a vertex $t \in T_R$ which both u and v can reach implying that there is a path between u and v after deleting X. Thus, any pair of vertices in T_L are connected after X is deleted.

Consider any arbitrary vertex $s \in V - T_L$. s has $2k$ vertex-disjoint paths to T_L so after deleting X there is a vertex $u \in T_L$ that it can reach. Thus, that after X is deleted, all vertices can reach T_L and T_L is connected implying that $G - X$ is connected giving a contradiction. ∎

A scheme in the AM model

In this subsection, we give a $[kn + h, v]$-scheme for k-vertex-connectivity using public randomness and achieve a smooth tradeoff too (similar to the promised-gap k-vertex-connectivity problem). This is the AM model in stream verification.

Lemma 9.6.17. *Under dynamic graph streams on n-node graphs, there exists an $[kn+h, v]$-AM-scheme for k-vertex-connectivity for any h, v such that $h \cdot v = n^2$.*

Proof. If the graph $G = (V, E)$ is not k-vertex-connected we can use the $[n + h, v]$-scheme of the simple algorithm where $h \cdot v = n^2$.

In the other case, we fix X as the lexicographically first minimum vertex cut (we do not know X but it is fixed). The prover now samples $\ell = 2 \log n$ vertices $t_1, t_2, \ldots, t_\ell$ uniformly at random using the public randomness. Use Corollary 9.6.12 to get a proof of k vertex-disjoint paths from t_i to every vertex in $V - \{t_i\}$ for all $i \in [\ell]$. This gives a $[kn + h, v]$-scheme.

We now prove the correctness of this scheme. If $|X| \geqslant 2k$, then we can use the proof of Lemma 9.6.15. Thus, $|X| < 2k$ implying $V - X$ has $\Omega(n)$ vertices (assuming $k < 0.49n$).

This means that with high probability, there is an $i^* \in [\ell]$ such that $t_{i^*} \in V - X$. Claim 9.6.3 shows that the graph is k-vertex-connected. This concludes the proof, and we have shown that the graph is k-vertex-connected using a $[kn + h, v]$-scheme for any h, v such that $h \cdot v = n^2$. ∎

9.6.3 Edge Connectivity Schemes

In this section, we discuss algorithms for k-edge-connectivity.

Theorem 23. *Under dynamic graph streams on n-node graphs, there exists a $[k^2 n + h, v]$-scheme for k-edge-connectivity for any h, v such that $h \cdot v = n^2$. In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(k^2 n)$.*

Theorem 24. *Under dynamic graph streams on n-node graphs, there exists an $[n, n]$-scheme for k-edge-connectivity (for any k). In particular, under such streams, the problem has non-trivial annotated streaming schemes with total cost $\widetilde{O}(n)$.*

Showing the graph is not k-edge-connected

Consider the case where the edge connectivity is $< k$. The prover sends the $k - 1$ edges whose deletion disconnects the graph and also sends one side S of this cut. The verifier first checks if the edges the prover sent were indeed part of the stream by running an $[h, v]$-scheme for subset check (Fact 9.3.7) for any h, v such that $h, v = n^2$. The verifier also has to check that there are no other edges between S and $T := V - S$. So the only allowed edges are within S, within T or the $k - 1$ edges sent by the prover. The verifier constructs a graph G' in a streaming fashion with all such possible edges and then checks if the edges in input graph are a subset of G'. This can be done using an $[h, v]$-scheme for subset check for any h, v such that $h \cdot v = n^2$ (Fact 9.3.7). This gives an $[n + h, v]$-scheme.

$[n, n]$-scheme for k-edge-connectivity

We now give the algorithm for when the edge connectivity is $\geqslant k$. We give a scheme independent of k that works for any k (this is basically solving minimum-cut). The idea is to simulate the 2-pass streaming algorithm for minimum-cut implied by [RSW18].

Proof of Theorem 24. The verifier when looking at the stream computes a $1 + \varepsilon$ cut sparsifier of the graph for $\varepsilon = 0.01$ and stores all vertex degrees. This can be done in space $\widetilde{O}(n/\varepsilon^2) = \widetilde{O}(n)$ [AGM12a].

The verifier then finds all the $1+\varepsilon$ approximate mincuts using the cut sparsifier. Compress all vertices that are on the same side of all of these cuts into supernodes. This graph of supernodes preserves all small cuts. The prover then sends the edges that go between the supernodes. [RSW18] proves that there are only $O(n)$ edges between the supernodes. So the verifier can store all of these edges and compute the minimum cut out of all these cuts. The verifier compares the size of this minimum-cut to the minimum degree and outputs the smaller of the two thus giving the exact mincut.

The verifier also needs to run a subset check (Fact 9.3.7) to check whether the edges the prover sent belong to the graph. This can be done using an $[n, n]$-scheme. Note that, the prover does not have to send all edges; just enough to make all cuts of size $\geqslant k$ so it is okay if the prover does not send all edges (thus a subset check is enough). ∎

Layering Algorithm

If the graph $G = (V, E)$ is not k-edge-connected, then we can use the proof of the simple algorithm which has an $[n + h, v]$-scheme.

Proof of Theorem 23. Fix an arbitrary vertex t. Use Claim 9.6.11 to get a proof of k edge-disjoint paths from t to every vertex in $V - \{t\}$. Thus, the proof size is $\widetilde{O}(h + k^2 n)$ words and can be verified using $\widetilde{O}(v)$ space for any h, v such that $h \cdot v = n^2$. This is a valid proof which shows that G is k-edge-connected. Say G has a cut S of size at most $k - 1$. Let $T := V - S$ be the other side of the cut containing t (wlog). There is a vertex $s \in S$ that has k edge-disjoint paths to t giving a contradiction. Thus, we get a $[h + k^2 n, v]$-scheme for k-edge-connectivity for any h, v such that $h \cdot v = n^2$. ∎

9.7 Annotated Streaming Schemes for SGT Streams

The k-connectivity algorithms for dynamic streams do not work for SGT streams because the verification fails when the input graph can have large frequencies for the edges. Also, the frequencies can be negative. So the proof using the layering lemma remains the same but the auxiliary information and verification changes. We prove the following algorithm for k-connectivity in SGT streams with parameter α:

Lemma 9.7.1. *Under SGT streams with parameter α, there exists a $[n^2 \log \alpha + k^2 n, 1]$-scheme for k-vertex-connectivity and k-edge-connectivity.*

Proof. For k-vertex-connectivity fix k arbitrary vertices $t_1, t_2, \ldots, t_k$. Use Corollary 9.6.12 to get a proof of k vertex-disjoint paths from t_i to every vertex in $V - \{t_i\}$ for all $i \in [k]$.

For k-edge-connectivity fix a vertex t and use Lemma 9.6.6 to et a proof for k edge-disjoint paths from t to every vertex in $V - \{t\}$. Thus, the proof sizes are $\widetilde{O}(k^2 n)$ words.

We now describe how the verification works. There are three things that need to be verified. First, all the edges that the prover sends should be a part of the input graph. Second, the edges sent by the prover should form paths. Third, the paths for a vertex should be vertex (edge) disjoint.

We can verify the second and third case like we did for dynamic streams. The prover will send the edges of the paths one path at a time, so it is easy to verify that the edges form a path. To verify the disjoint paths the prover sends the vertices (edges) he is going to use in the proof in sorted order and then sends the proof. This only increases the proof size by a constant factor. It is easy to verify that the vertices (edges) arriving in sorted order are distinct in $\widetilde{O}(1)$ space. The only thing left to do is check if the vertices (edges) sent upfront are the same as the ones used in the proof of disjoint paths. We do this by running an equality check using the ℓ_0-sampler trick using $\widetilde{O}(1)$ space (Proposition 9.3.11). When we get the vertices upfront we add them to an ℓ_0-sampler and when we see them in the proof we subtract them from the sampler. If there is nothing left in the sampler in the end, then the two multi-sets of vertices are identical. Since the first one had distinct elements, so did the second one. The blow-up in proof size is at most a factor of 2 because the auxiliary information just contains the vertices used in the proof. Also, the space on the verifier side is $\widetilde{O}(1)$ words for each vertex, but this space can be reused and thus, it fits in our budget of $\widetilde{O}(1)$ words.

We now come to the hardest part which is verifying that the edges used in the proof indeed belong to the input graph. The prover initially sends a list of all edges of the input graph along with their multiplicities sorted by sign (first sends all the positive frequency edges and then the negative frequency edges). For technical reasons, the prover scales the multiplicities by n^2. It is easy to do a multi-set equality check for these edges with multiplicities sent upfront and the edges from the input stream using the ℓ_0-sampler trick in $\widetilde{O}(1)$ verification space (Proposition 9.3.11).

In parallel, we create a sampler for the positive edges and another for the negative edges and insert the edges the prover sent upfront into the appropriate sampler according to the sign. During the proof, we ask the prover to also mention the sign of each edge and we subtract it from the appropriate sampler. This increases the proof size by a constant factor. After the proof, the prover sends a list of all the remaining edges i.e. edges not used in the proof along with their residual multiplicities (the residual multiplicities take the scaling by n^2 into account). We use ℓ_0-sampler trick (Proposition 9.3.11) to check the equality of the multi-sets, the set of positive edges send upfront and the positive edges

used in the proof plus the residual positive edges using $\widetilde{O}(1)$ space. We do the same for the negative edges.

The correctness is as follows. The prover cannot lie about the signs or multiplicities when the edges are sent upfront because we run an equality check with the input stream. If the prover ever lies about a sign during the proof (or in while sending the residual edges) then the edge gets subtracted from the wrong sampler. This sampler has a negative frequency on the coordinate corresponding to that edge which can only decrease later (because we are only subtracting edges) implying that the sampler check will fail.

If the prover lies about the multiplicities in the residual edges then the equality check will fail. Thus, we just have two separate equality check problems. The scaling by n^2 ensures that an edge can be used multiple times during the proof (it can be used at most $kn \leqslant n^2$ times). Ignoring the residual multiplicities we are just performing a subset check.

The space used by the verifier is just $\widetilde{O}(1)$ words. The proof size, ignoring the edges sent upfront and the residual edges, is $\widetilde{O}(k^2 n)$. The edges sent upfront and the residual edges take size $O(n^2 \log(n^2 \alpha)) = \widetilde{O}(n^2 \log \alpha)$. The probability of failure is at most $1/\text{poly}(n)$ (by union bounding over the polynomially many failure probabilities of $1/\text{poly}(n)$). Therefore, the proof along with the auxiliary information takes size $\widetilde{O}(n^2 \log \alpha + k^2 n)$ and can be verified in $\widetilde{O}(1)$ space, proving the lemma. ∎

Chapter 10

Sublinear-Time Non-Adaptive Matching Size

10.1 Introduction

Maximum matching is among the most fundamental and well-studied problems in combinatorial optimization and theoretical computer science. A **matching** in a graph is a set of edges no two of which share an endpoint, and a *maximum matching* is a matching of the largest size. Edmonds, in his seminal paper [Edm65], gave the first polynomial-time algorithm for the maximum matching problem, a breakthrough that influenced the formal development of computational complexity theory and inspired later formalizations of complexity classes, such as P and NP. This runtime was later improved to $O(m\sqrt{n})$ [HK73, MV80] where n is the number of vertices and m is the number of edges. Further improving this runtime is a major open problem. The recent breakthrough on max flow [CKL+22, VDBCK+23] implies an $m^{1+o(1)}$ time deterministic algorithm for bipartite matching. While $O(m\sqrt{n})$ remains the best-known bound for exact solutions in general graphs, allowing approximations makes the problem significantly easier. For instance, a greedy maximal matching gives a 2-approximation in linear time. A $(1+\varepsilon)$-approximation to the maximum matching can be achieved in $O(m/\varepsilon)$ time [HK73, DP14].

However, in the era of massive graphs, the traditional goal of linear-time computation becomes inadequate. This motivates the study of *sublinear-time* algorithms: algorithms that estimate the solution without reading the entire input. In the case of maximum matching, a natural question that arises is whether there exist sublinear-time algorithms for approximating the maximum matching.

10.1.1 Prior Work

The sublinear-time setting assumes restricted access to the graph, typically via the *adjacency list model* (our focus) or the *adjacency matrix model*. In the former, algorithms may query the degree of a vertex or its i^{th} neighbor (the algorithm receives NULL if the vertex has fewer than i neighbors). In the latter, the algorithm specifies a pair of vertices and learns whether they are adjacent.

Unfortunately, there are lower bounds that answer the aforementioned question on sublinear-time approximate matching in the negative. In the adjacency matrix model or the adjacency list model, any algorithm that outputs a constant approximate matching needs $\Omega(n^2)$ queries [ACK19b, BKS23b]. While this lower bound rules out the possibility of efficiently outputting the edges of a matching, it does not preclude meaningful progress. Rather than retrieving the matching itself, we can shift our focus to estimating its size, which turns out to be a much more feasible goal.

The first wave of results [PR07, NO08, YYI09, ORRR12] focused mainly on bounded-degree graphs and were not sublinear-time for maximum degree $\Delta = \Theta(n)$. The second wave of results run in truly sublinear-time even on graphs with a large maximum degree, starting with the works [CKK20, KMNFT20]. [Beh22] marked a milestone by establishing a 2-approximation in $\widetilde{O}(n)$ time[1]. These were followed by a series of results by [BRRS23, BRR23, BKS23b, BRR23] culminating in a $(1, \varepsilon n)$-approximation[2] in $o(n^2)$ time [BKS23a]. Note that all these results hold in the adjacency list query model, except for the last one, which holds in the adjacency matrix model but is believed to extend to the adjacency list setting as well [BRR24].

These sublinear-time upper bounds are complemented by lower bounds on the query complexity which bound the running time of any algorithm that approximates the maximum matching size. These begin with the $\Omega(n)$ lower bound for constant-factor approximations [PR07]. Subsequent results established super-linear lower bounds for achieving better approximations, including $\Omega(n^{1.2-o(1)})$ for a $(1.5-\varepsilon)$-approximation [BRR23] and $\Omega(n^{2-f(\varepsilon)})$ for a $(1, \varepsilon n)$-approximation [BRR24].

While these results have significantly advanced our understanding of sublinear-time algorithms for maximum matching, they all rely on adaptive querying strategies, where queries are chosen based on answers to previous queries. In many cases, non-adaptive queries offer distinct advantages. Since all queries come from a predetermined distribution, they can be executed independently, making them well-suited for parallel computation

[1] Throughout the paper, we use $\widetilde{O}(f) := O(f \cdot \text{polylog}\,(n))$ to hide poly-logarithmic factors in n.

[2] An (x, y)-approximation achieves multiplicative error x and additive error y.

frameworks. This motivates the study of algorithms for maximum matching in a fully non-adaptive setting, where the distribution over queries must be fixed in advance.

Non-adaptive query models have been widely studied in property testing algorithms [GT01, BSHR03, Fis04b, GR11, BS23], often revealing fundamental gaps between adaptive and non-adaptive complexities [RS06, GR07, CG17, GW22]. For instance, [RS06] shows that in the bounded degree model for graph property testing, adaptivity is essential. This motivates understanding whether adaptivity is necessary for approximating the maximum matching size in sublinear time.

However, for approximating maximum matching size, no prior work has explored the power or limitations of non-adaptive queries. Investigating this model could provide new insights into the role of adaptivity in sublinear-time graph algorithms and reveal whether adaptivity is essential for achieving strong approximations. Thus, similar to the adaptive case [Beh22], we would ideally like to obtain a constant-factor approximation around 2 using $\widetilde{O}(n)$ non-adaptive queries. However, since this may be challenging, we instead consider a more relaxed question:

> *Can we get an $O(1)$ or even an $\widetilde{O}(1)$-approximation to the maximum matching size using $\widetilde{O}(n)$ non-adaptive adjacency list queries?*

10.1.2 Our Contributions

We answer this question in the negative by proving the following theorem:

Theorem 29. *For any constant $\gamma > 0$ there exists a constant $\varepsilon > 0$ such that any randomized algorithm that is given access to an arbitrary graph using non-adaptive adjacency list queries and can get an $n^{1/3-\gamma}$-approximation to the maximum matching size with probability at least $2/3$, needs to use $\Omega(n^{1+\varepsilon})$ queries.*

Note that our hard instance is a multi-graph; we allow multiple edges to simplify the analysis, though we believe similar bounds hold without them.

This theorem rules out any $\widetilde{O}(1)$-approximations in $\widetilde{O}(n)$ queries. In fact, it says something much stronger. It rules out even a polynomial approximation better than $n^{1/3}$ in $\widetilde{O}(n)$ queries. We prove a more general theorem which we state below:

Theorem 30. *Any randomized algorithm that is given access to an arbitrary graph using non-adaptive adjacency list queries and can get an n^{δ}-approximation to the maximum matching size with probability at least $2/3$, needs to use $\Omega(n^{1+\varepsilon})$ queries for any $\varepsilon, \delta > 0$ satisfying $n^{2\varepsilon+3\delta} \log^3 n = o(n)$.*

This theorem is more general and gives a tradeoff between the approximation factor and the number of queries required. Moreover, since our hard instance is a bipartite graph, the lower bound also applies to algorithms restricted to bipartite inputs. If we want to stick to an $\widetilde{O}(1)$-approximation, then Theorem 30 shows that we get a strong lower bound of $\Omega(n^{1.5-o(1)})$ queries. In particular, we have the following corollary.

Corollary 10.1.1. *Any randomized algorithm that is given access to an arbitrary graph using non-adaptive adjacency list queries and can get better than an $\widetilde{O}(1)$-approximation to the maximum matching size with probability at least $2/3$, needs to use $\Omega(n^{1.5-o(1)})$ queries.*

These results highlight the necessity of adaptivity in existing algorithms. In the $\widetilde{O}(n)$ query regime, there is a drastic difference: adaptive algorithms achieve a 2-approximation, while non-adaptive ones cannot get better than an $n^{1/3}$ approximation. But is anything achievable with non-adaptive queries? At first glance, the model may seem too restrictive to yield meaningful approximations. We show that this is not the case by presenting an algorithm that achieves an n^{δ}-approximation to the maximum matching size using $O(n^{2-2\delta}\log n)$ non-adaptive adjacency list queries. In particular, when $\delta = 1/2$, the algorithm gives a $\sqrt{n}$-approximation in $O(n \log n)$ queries. This shows that non-adaptive queries, despite their limitations, can still achieve non-trivial approximations.

Theorem 31. *There is a randomized algorithm that with probability $1 - 1/n$ can output an n^{δ}-approximation to the matching size using $O(n^{2-2\delta}\log n)$ non-adaptive queries in the adjacency list model for any constant $0 < \delta < 1$. In particular, when $\delta = 1/2$, the algorithm gives a $\sqrt{n}$-approximation to the matching size in $O(n \log n)$ queries.*

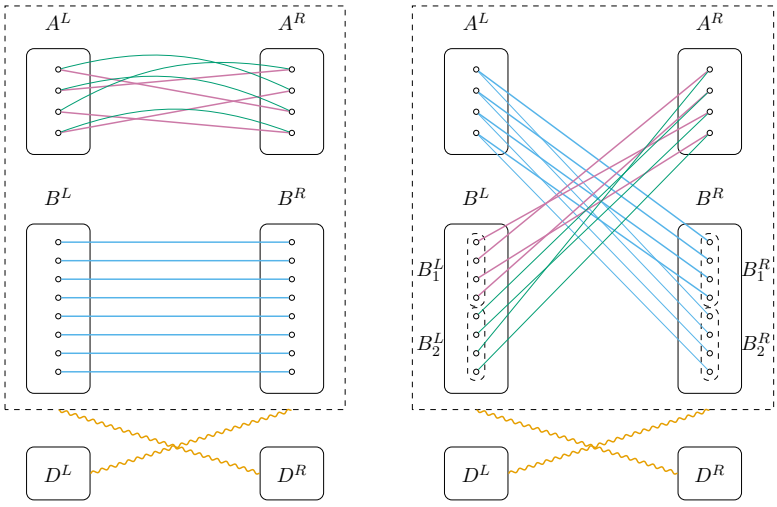
If we fix the number of queries to be $\widetilde{O}(n)$, then our algorithm achieves a $\sqrt{n}$-approximation. On the other hand, Theorem 29 shows that any algorithm achieving an $n^{1/3-\delta}$-approximation requires more than $\widetilde{O}(n)$ queries. Together, these results provide polynomial upper and lower bounds in the $\widetilde{O}(n)$ query regime.

10.1.3 Technical Overview

In this subsection, we present a simplified construction and analysis to build intuition for a weaker version of our lower bound.

Graph Construction. Consider the construction illustrated in Figure 10.1. Note that this is not the final distribution. We have a bipartite graph $G = (V = L \cup R, E)$, where the vertex set is partitioned into three subsets: A, B, and D, of sizes $2n^{1-\delta}$, $2n$, and $2n^{1-\delta}$, respectively. Each of these sets is evenly split between the left and right sides of

the bipartition. The vertices in D are **dummy** nodes whose sole purpose is to increase the degrees of vertices in $A \cup B$, which we refer to as the **core**.



(a) Yes Instance (b) No Instance

Figure 10.1: Hard Distribution

In the *yes* case, there is a perfect matching of size n between the B vertices, and n^δ random matchings between the A vertices. In the *no* case, the B vertices are divided into groups of size $n^{1-\delta}$, and each group has a random perfect matching to A. In both cases, every vertex in the core has an edge to a vertex in D on the opposite side with probability $1/2$. This completes the construction. The adjacency lists of all vertices are randomly permuted. Due to the edges to dummy vertices, the degree of each core vertex is $\Omega(n^{1-\delta})$.

It is easy to see that in the *yes* case, the matching has size at least n, while in the *no* case, the set $A \cup D$ forms a vertex cover of the graph, implying that the maximum matching has size at most $4n^{1-\delta}$. We now fix a deterministic algorithm Alg that makes $q = n^{1+\varepsilon}$ non-adaptive queries and distinguishes between the *yes* and *no* cases. To hide the structure of the edges within the core, we apply a random permutation to the names of all vertices, so that Alg has no information about which vertex belongs to which set. The queries made by Alg are distributed arbitrarily across the vertex set, but the random permutation ensures that, on average, each vertex is queried approximately n^ε times.

Disjoint union of stars. We define a vertex as **happy** if it discovers an edge within the core. For a single query, this happens with probability at most $n^\delta/\Omega(n^{1-\delta})$. Leveraging the randomness introduced by the permutation that shuffles vertex names, and noting that

the algorithm Alg makes an average of n^{ε} queries, the overall probability that a vertex is happy becomes $O(n^{2\delta+\varepsilon-1})$. Therefore, the expected number of happy vertices is $O(n^{2\delta+\varepsilon})$. Now consider an edge: the probability that both its endpoints are happy is $O(n^{4\delta+2\varepsilon-2})$. Since there are just $2n$ edges in the core, the number of edges with both happy endpoints will be 0 with a large probability. As a result, the algorithm Alg observes only a disjoint union of stars in the core—since the petals of the stars are not happy and cannot discover neighbors in the core.

Note that if a star has degree greater than 1, then its center must be an A vertex, since B vertices participate in at most one matching in the *yes* case. However, if a star has degree exactly 1, its center could be either an A or a B vertex. We can tell Alg the labels of the star centers—whether they belong to A or B—but what remains crucially hidden is the labels of the petals. In the *yes* case, all A star centers are connected to A petals, and all B star centers to B petals. In the *no* case, the pattern is reversed: A star centers have B petals, and B star centers have A petals. To show that Alg cannot distinguish between the *yes* and *no* cases, it suffices to prove that Alg succeeds with probability at most $1/2 + o(1)$.

Coupling Argument. We formalize this by constructing a coupling between the *yes* and *no* distributions. The coupling is a one-to-one mapping from the *no* distribution to the *yes* distribution such that the observed graph (the edges seen by Alg) remains identical under this mapping. To construct this mapping, we partition the support of both distributions into blocks, where all elements within a block correspond to the same observed graph. We then define the coupling to map between corresponding blocks in the two distributions. Finally, we show that this mapping covers a $(1-o(1))$ fraction of the support in both the *yes* and *no* distributions. Thus, for most observed graphs, the probability that they arise from the *yes* or *no* distribution is roughly $1/2$. For the remaining $o(1)$ fraction, the algorithm Alg can succeed with probability at most 1, leading to an overall success probability of at most $1/2 + o(1)$. This coupling argument is the main technical ingredient of our proof.

10.1.4 Related Work

Comparison with [BRR23]. Behnezhad, Roghani, and Rubinstein (henceforth BRR23) prove an $\Omega(n^{1.2-o(1)})$ lower bound for a $(2/3+\varepsilon)$-approximation to the maximum matching size using adaptive queries in the adjacency list model. Our setting is non-adaptive, and our lower bound exhibits a significantly larger gap between the matching sizes in the yes and no distributions.

Both works use coupling-based arguments, but have different structural properties of the observed graph. BRR23 shows that their observed graph forms a rooted forest, while

we prove that ours forms a disjoint union of stars. The main technical part of their proof is a correlation decay, but we do not need that in our proof because we get a *disjoint* union of stars. However, given the large approximation parameter, we need a fundamentally different coupling argument.

Finally, while the models differ, the $n^{1.2}$ barrier in their work does not arise in our setting, and we achieve a better exponent: we show that even polylogarithmic approximations require $\Omega(n^{1.5-o(1)})$ queries.

Other Related Work. In addition to sublinear-time algorithms, maximum matching has been extensively studied in several related computational models. Maximum matching is one of the most central problems in the *dynamic graph* literature [OR10, GP13, BGS15, BS16, BK22, Beh23, BKSW24, ABKL23, BKS23a, Liu24, BG24, AS25]. Some of these works have shown that sublinear-time matching size estimators can be used to design dynamic algorithms.

The problem has also been heavily explored in the *streaming model*, including insertion-only streams under both single-pass [FKM+04, ABKL23] and multi-pass settings [FMU22, KN24], as well as dynamic streams, again in both single-pass [AKLY16, CCE+16, DK20, AS22] and multi-pass settings [AG18, Ass24, ABK+24]. Finally, the maximum matching problem has also been studied in *distributed models* [Fis20, FMU22, FGG+23, GG23, GT24, MS25].

10.2 Preliminaries

Graph Notation and Terminology. For a graph $G = (V, E)$, we use n to represent a constant factor approximation to the number of vertices. m represents the number of edges and $\mu(G)$ to represent the size of any *maximum matching* of G. We use $\deg(v)$ and $N(v)$ for each vertex $v \in V$ to denote the degree and neighborhood of v, respectively. Any vertex v with $\deg(v) = 0$ is called an **isolated** vertex. A **star** graph is one where there is a central vertex, called the **star center**, connected to all other vertices, called **petals**, with no edges between the petals.

For a subset F of edges in E, we use $V(F)$ to denote the vertices incident on F; similarly, for a set U of vertices, $E(U)$ denotes the edges incident on U. We further use $G[U]$ for any set U of vertices to denote the induced subgraph of G on U. We use $|T|$ to denote the number of elements in T when it is a set, and the size of its support when T is a distribution.

A **perfect matching** between two sets of vertices L and R is a matching in which each edge connects a unique vertex in L to a unique vertex in R, and every vertex in $L \cup R$ is

Symbol	Description
Graph Components	
$G = (V, E)$	Bipartite graph with $V := L \cup R$
A, B, D	Sets of vertices $A^L \cup A^R$, $B^L \cup B^R$, $D^L \cup D^R$ of sizes $2n^{1-\delta}, 2n, 2n^{1-\delta}$
C	$C := A \cup B$ is the core of G
B_i^L, B_i^R	Groups within B^L and B^R ($i \in [n^\delta]$)
$b_{i,j}^R$	j^{th} vertex in group B_i^R ($j \in [n^{1-\delta}]$)
a_i^R	i^{th} vertex in A^R ($i \in [n^{1-\delta}]$)
b_i^R	i^{th} vertex in B^R ($i \in [n]$)
$\mathcal{U}, \mathcal{V}$	Label sets for L and R
Parameters	
n'	Size of L and R, $n' = \Theta(n)$
δ	Parameter for approximation factor
ε	Parameter for number of queries
$q := n^{1+\varepsilon}$	Total number of non-adaptive queries
q_v	Number of queries for vertex v
d^*	Degree of each vertex in C

Table 10.1: Notation for Graph Components and Parameters.

Symbol	Description
Distributions	
$\mathcal{D}_{\text{YES}}$, $\mathcal{D}_{\text{NO}}$	"Yes" and "No" input distributions
$\mathcal{D}$	Mixture: $\frac{1}{2}\mathcal{D}_{\text{YES}} + \frac{1}{2}\mathcal{D}_{\text{NO}}$
$\mathcal{D}^D$	Distribution of edges between Core and D
$\mathcal{D}^C_{\text{YES}}$, $\mathcal{D}^C_{\text{NO}}$	Distribution of edges in the Core in $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$
Random Variables	
π	Random labeling permutation from Π
$\ell(v)$	Adjacency list of vertex v
$\psi(v)$	Randomly chooses core indices in $\ell(v)$
$\sigma_C(v), \sigma_D(v)$	Random permutation of core and dummy neighbors
$\sigma(v)$	Random permutation of all neighbors for $v \in D$
Observed Graph and Algorithm	
Alg	Fixed deterministic algorithm making non-adaptive queries
$G_{\text{obs}} = (V, E_{\text{obs}})$	Directed graph observed by Alg
E^C_{obs}	Observed edges in core (C)

Table 10.2: Notation for the Distributions, Randomness, and Observed Graph.

matched. This requires that $|L| = |R|$. An **identity matching** between L and R assumes that both sets are ordered, and matches the i^{th} vertex in L to the i^{th} vertex in R for all $i \in [|L|]$.

Let M^* denote a maximum matching in the graph. We define an n^δ-**approximate matching** as a matching of size at least a $1/n^\delta$ fraction of the size of M^*. We define an n^δ-**approximation** to the *matching size* as an estimate s such that

$$n^{-\delta} \cdot |M^*| \leqslant s \leqslant |M^*| .$$

Probability Bounds. Throughout, we will use the term "**with high probability**" to mean with probability at least $1 - o(1)$. In some places, we obtain a probability bound of $1 - 1/\text{poly}(n)$. We do not refer to this as occurring "with high probability"; instead, we state the bound explicitly.

10.3 The Lower Bound

In this section, we show that any algorithm that can get an $O(n^\delta)$-approximation to the maximum matching size, needs to use $\Omega(n^{1+\varepsilon})$ non-adaptive queries such that $n^{2\varepsilon+3\delta} \log^3 n = o(n)$.

Theorem 30. *Any randomized algorithm that is given access to an arbitrary graph using non-adaptive adjacency list queries and can get an n^δ-approximation to the maximum matching size with probability at least $2/3$, needs to use $\Omega(n^{1+\varepsilon})$ queries for any $\varepsilon, \delta > 0$ satisfying $n^{2\varepsilon+3\delta}\log^3 n = o(n)$.*

We will prove this theorem in this section. The goal is to establish a lower bound for randomized algorithms that achieve an $O(n^\delta)$-approximation to the maximum matching size. To accomplish this, we construct a hard input distribution under which no deterministic algorithm, restricted to $n^{1+\varepsilon}$ queries, can guarantee such an approximation. By Yao's minimax principle, this immediately yields a worst-case lower bound for randomized algorithms. The construction is shown below.

10.3.1 The Hard Instance Construction

The hard instance is a bipartite graph $G = (V, E)$ (see Figure 10.2).

The vertex set: The vertex set on the left L is of size $n' = n + 2n^{1-\delta}$ and is composed of three distinct subsets A^L, B^L, D^L of size $n^{1-\delta}, n$ and $n^{1-\delta}$ respectively. Similarly, the vertex set on the right R of size n' is composed of A^R, B^R, D^R of size $n^{1-\delta}, n$ and $n^{1-\delta}$ respectively. So the number of vertices in the graph is $2n' = \Theta(n)$. We also divide B^L (similar for B^R) into groups B_i^L for $i \in [n^\delta]$ of size $n^{1-\delta}$. Throughout, we may write A, B, D to respectively refer to sets $A^L \cup A^R, B^L \cup B^R, D^L \cup D^R$. We use a_i^R to denote the i^{th} vertex of A^R. We do the same for the others. Additionally, we use $b_{i,j}^R$ to denote the j^{th} vertex in the group B_i^R. Note that when considering vertices, capital letters denote sets of vertices and lowercase letters denote vertices. Finally, we will refer to $C := A \cup B$ as the **core** and call vertices in D as **dummy** vertices (since they are added to just raise the degrees of all vertices in the core).

The edge set: For the edge set E, we give two distributions: in distribution $\mathcal{D}_{\text{YES}}$ the maximum matching of G is sufficiently large, and in distribution $\mathcal{D}_{\text{NO}}$ the maximum matching of G is sufficiently small. The final input distribution $\mathcal{D} := \frac{1}{2}\mathcal{D}_{\text{YES}} + \frac{1}{2}\mathcal{D}_{\text{NO}}$ draws its input from $\mathcal{D}_{\text{YES}}$ with probability $1/2$ and from $\mathcal{D}_{\text{NO}}$ with probability $1/2$. The following edges are common in both distributions $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ and come from distribution $\mathcal{D}^D$:

- Every vertex in A^R has $n^{1-\delta}/2 - n^\delta$ edges to random vertices in D^L sampled without replacement (same between A^L and D^R).
- Every vertex in B^R has $n^{1-\delta}/2 - 1$ edges to random vertices in D^L sampled without replacement (same between B^L and D^R).

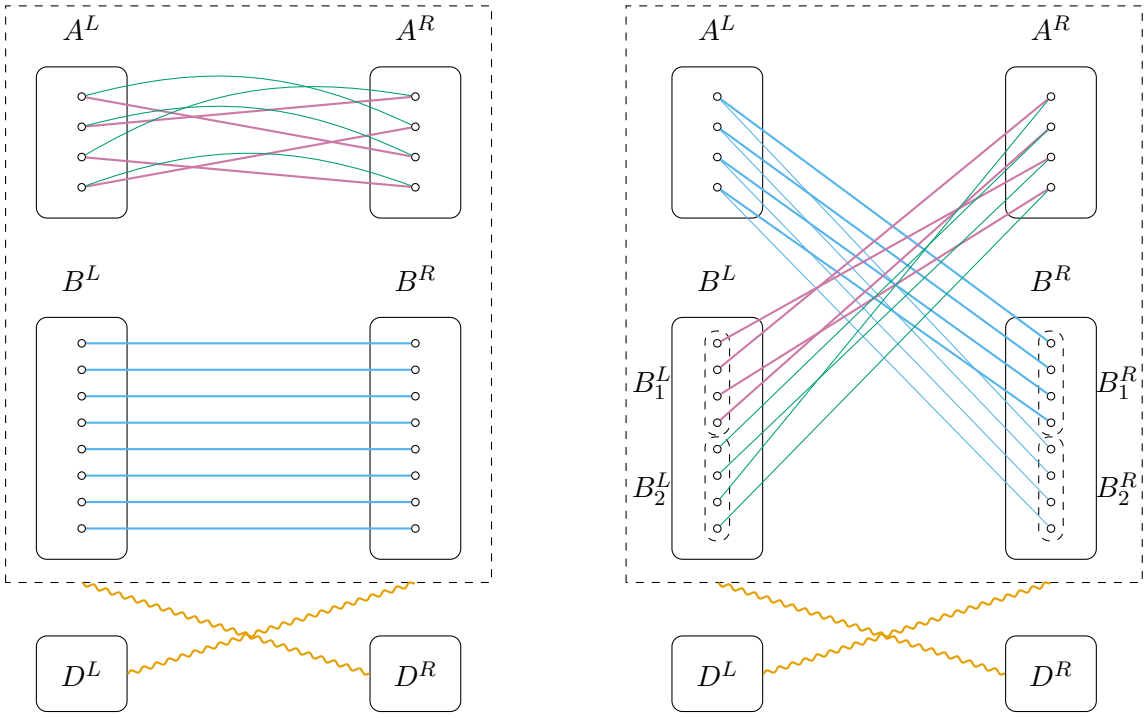
The following edges are specific to $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ respectively:

- In $\mathcal{D}_{\text{YES}}$, we additionally add an identity matching between B^L and B^R i.e. edges (b_i^L, b_i^R) for all $i \in [n]$. We also add n^δ random perfect matchings $M_1, M_2, \ldots, M_{n^\delta}$ between A^L and A^R. We call this entire distribution in the core unique to $\mathcal{D}_{\text{YES}}$, $\mathcal{D}_{\text{YES}}^C$.
- In $\mathcal{D}_{\text{NO}}$, we add a random perfect matching between each group B_i^L of B^L and A^R for $i \in [n^\delta]$. We also add an identity matching between each group B_i^R of B^R and A^L for $i \in [n^\delta]$. This means that for every group B_i^R we add edges $(b_{i,j}^R, a_j^L)$ for $j \in [n^{1-\delta}]$. We call this entire distribution in the core unique to $\mathcal{D}_{\text{NO}}$, $\mathcal{D}_{\text{NO}}^C$.

This concludes the construction of the yes and no distributions $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$. We note that this construction gives us a multi-graph. The edges in the core will be called **core edges**. We will use the term A-to-A edges to denote the edges between A^L and A^R (same for B-to-B and A-to-B edges). For a vertex in the core, a neighbor in the core is called a **core neighbor** and its index in the adjacency list is called a **core index**. We also emphasize that we pick a random **labeling permutation** π over the vertices to give a random label to each vertex which will be seen by the algorithm in the adjacency list. The vertices have labels $\mathcal{U} := \{u_1, u_2, \ldots, u_{n'}\}$ for the L side and $\mathcal{V} := \{v_1, v_2, \ldots, v_{n'}\}$ for the R side. The set of all labeling permutations is Π.

We now describe the **adjacency lists** of these vertices. Note that the vertices will not have their original names in the structure but the labels given to them by the labeling permutation $\pi \in \Pi$. We let $\ell(v)$ be the random variable denoting the adjacency list of vertex $v \in V$. The adjacency list of every vertex should contain its neighbors in random order. Thus, the randomness we need to define for the adjacency lists is for picking the locations of the core indices and random permutations over the elements that go in the core and the dummy indices.

For $v \in C$, let $\psi(v)$ be the random variable which picks the core indices uniformly at random out of all the indices in the adjacency list of v. For $v \in C$, let $\sigma_C(v)$ be the random variable which picks a uniformly random permutation over the core neighbors of v. For $v \in C$, let $\sigma_D(v)$ be the random variable which picks a uniformly random permutation over the dummy neighbors of v. Note that for $v \in B$, $\sigma_C(v)$ is always the identity permutation because there is just one core index. Therefore, for vertices $v \in C$, $\psi(v), \sigma_C(v)$ and $\sigma_D(v)$ together determine the positions and order of neighbors in $\ell(v)$. For vertices $v \in D$, we just have a uniformly random permutation over all the neighbors in the adjacency list which is given by $\sigma(v)$.



(a) Yes Instance ($\mathcal{D}_{\text{YES}}$)

(b) No Instance ($\mathcal{D}_{\text{NO}}$)

Figure 10.2: Illustration of a Yes and a No instance. Blue edges denote identity matchings: a matching of size 8 in the Yes case, and two matchings of size 4 each in the No case. Pink and Green edges form matchings that are structurally identical across the instances but connect to different vertex sets on the left. Orange squiggly lines represent edges from D^L to $A^R \cup B^R$ (same for D^R), existing with probability roughly 1/2.

Finally, we note that we give away the bipartition L, R of the graph for free before the queries are made. An algorithm that makes $\widetilde{O}(n)$ queries can figure out the sets D, C any vertex belongs to after the non-adaptive queries are made by looking at the approximate degree of every vertex. What is crucially hidden from the algorithm, however, is whether a vertex $v \in C$ belongs to A or B.

We fix a deterministic algorithm Alg that makes a fixed number of non-adaptive adjacency list queries $q = n^{1+\varepsilon}$ and differentiates between $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$. For each vertex, the queries Alg makes for that vertex are fixed. The number of queries Alg makes is q_i^L for vertex u_i and q_j^R for vertex v_j. If we consider some vertex $v \in V$ then we let q_v be the number of queries Alg makes to the adjacency list of v. When a query on a vertex say u

returns an edge, we think of the edge as directed going out of u for analysis purposes. We now formalize this by defining the observed graph $G_{\text{obs}} = (V, E_{\text{obs}})$:

- The graph is directed with vertex set V.
- For every vertex $v \in V$, all the edges discovered by the queries on v belong to the edge set E_{obs} and are directed out of v.
- Each directed edge out of v is labeled with its position in the adjacency list of v.

We also define $E^C_{\text{obs}} := E_{\text{obs}}(C)$ as the set of edges in the core and $E^D_{\text{obs}} := E_{\text{obs}} \setminus E^C_{\text{obs}}$ as the edges between the core and D.

10.3.2 Properties of the Hard Instance

In this subsection, we observe some simple properties of the hard instance. We start with the following bounds on vertex degrees which follow immediately from the construction.

Observation 10.3.1. For any graph G drawn from $\mathcal{D}_{\text{YES}}$ or $\mathcal{D}_{\text{NO}}$, the degree of any vertex in C is $d^* := n^{1-\delta}/2$.

Notice that the degrees of vertices in A and B are the same, and it is difficult for algorithm Alg to tell these two apart with a few number of queries. Vertices in A have $n^{1-\delta}/2 - n^{\delta}$ neighbors in D and n^{δ} neighbors in the core. Vertices in B have $n^{1-\delta}/2 - 1$ neighbors in D and 1 neighbor in the core. However, it is easy for Alg to identify D vertices by just querying the degrees. This is because the degrees of the vertices in D are very concentrated around their expectation.

Claim 10.3.2. *For all $v \in D$, $\mathbb{E}[\deg(v)] = n/2 + n^{1-\delta}/2 - 2n^{\delta}$ and $\deg(v) \in (1 \pm 0.2) \cdot (n/2)$ with high probability.*

Proof. Let $v \in D$ be a vertex and let X_i for $i \in [n^{1-\delta}]$ be 1 if a_i has an edge to v and 0 otherwise. Let Y_i for $i \in [n]$ be 1 if b_i has an edge to v and 0 otherwise. Note that if v is on the left then it only has edges to vertices on the right and vice versa, so we drop the superscript of L and R.

We have $\mathbb{E}[X_i] = \Pr(X_i = 1) = (d^* - n^{\delta}/n^{1-\delta}) = 1/2 - n^{2\delta-1}$. Let $X = \sum_i X_i$ be the number of edges v has to A. We have $\mathbb{E}[X] = n^{1-\delta} \cdot (1/2 - n^{2\delta-1}) = n^{1-\delta}/2 - n^{\delta}$. Similarly, we have $\mathbb{E}[Y_i] = \Pr(Y_i = 1) = (d^* - 1/n^{1-\delta}) = 1/2 - n^{\delta-1}$. Let $Y = \sum_i Y_i$ be the number of edges v has to B. We have $\mathbb{E}[Y] = n \cdot (1/2 - n^{\delta-1}) = n/2 - n^{\delta}$.

Then $Z = X + Y$ represents the degree of v. We have $\mathbb{E}[Z] = \mathbb{E}[X] + \mathbb{E}[Y] = n/2 + n^{1-\delta}/2 - 2n^{\delta}$. We now show concentration on Z by showing concentration on X and Y. X and Y are sums of independent random variables (since the choices across vertices are independent), so we can use the Chernoff bound (Proposition 2.4.3) to prove concentration for them:

$$\Pr(|X - \mathbb{E}[X]| > 0.1\,\mathbb{E}[X]) \leqslant 2\exp(-(0.1)^2\,\mathbb{E}[X]/3) \ll 1/\mathrm{poly}(n),$$

$$\Pr(|Y - \mathbb{E}[Y]| > 0.1\,\mathbb{E}[Y]) \leqslant 2\exp(-(0.1)^2\,\mathbb{E}[Y]/3) \ll 1/\mathrm{poly}(n).$$

Thus, we get that $Z \in (1 \pm 0.1)\,\mathbb{E}[Z]$ with probability $1 - 1/\mathrm{poly}(n)$. We union bound over all vertices and relax the bound slightly to get the bound in the claim. ∎

We now look at the graphs in $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ and show that the matching sizes in both cases are very different.

Lemma 10.3.3. *Let $G_{YES} \sim \mathcal{D}_{YES}$ and $G_{NO} \sim \mathcal{D}_{NO}$. Then it holds with probability 1 that $|\mu(G_{YES})| \geqslant n$ and $|\mu(G_{NO})| \leqslant 4n^{1-\delta}$.*

Proof. All graphs in $\mathcal{D}_{\text{YES}}$ have a matching of size n between the B vertices. The set $A \cup D$ forms a vertex cover for all graphs in $\mathcal{D}_{\text{NO}}$ implying that the matching size is at most $4n^{1-\delta}$ (the size of the maximum matching and minimum vertex cover are the same in bipartite graphs). ∎

Although the matching sizes in the yes and no instances differ significantly, we aim to show that since Alg makes a limited number of queries, it cannot distinguish between them with probability better than $\frac{1}{2} + o(1)$, which is only slightly better than random guessing. The central lemma we will establish is the following:

Lemma 10.3.4. *The success probability of* Alg *in distinguishing between $\mathcal{D}_{NO}$ and $\mathcal{D}_{YES}$ using $q = n^{1+\varepsilon}$ queries is at most $1/2 + o(1)$. This holds under the constraint $n^{2\varepsilon+3\delta}\log^3 n = o(n)$.*

With Lemma 10.3.4 in hand, we can now immediately prove Theorem 30.

Proof of Theorem 30. Assume towards a contradiction that there is a randomized algorithm that uses $o(n^{1+\varepsilon})$ non-adaptive adjacency list queries and gives better than a $n^{\delta}/4$ approximation to the maximum matching size with probability at least $2/3$.

We know by Yao's minimax principle [Yao77] that distributional query complexity for any distribution is a lower bound on randomized query complexity. So consider the distribution $\mathcal{D} = \frac{1}{2}\mathcal{D}_{\text{YES}} + \frac{1}{2}\mathcal{D}_{\text{NO}}$. The existence of such an algorithm implies that there is a deterministic algorithm Alg which when given an input from $\mathcal{D}$, gives better than a $n^{\delta}/4$ approximation to the maximum matching size with probability at least $2/3$.

We also know by Lemma 10.3.3 that every graph in $\mathcal{D}_{\text{YES}}$ has a matching size of at least n and the size of the maximum matching for every graph in $\mathcal{D}_{\text{NO}}$ is at most $4n/n^{\delta}$. Thus, any algorithm that can approximate the maximum matching size with a factor better than $n^{\delta}/4$ with probability at least $2/3$ can differentiate between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ with probability at least $2/3$. But we know by Lemma 10.3.4 that Alg makes q queries and distinguishes between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ with success probability at most $1/2+o(1)$. This gives us a contradiction because we got a deterministic algorithm with probability of success $2/3$. Thus, we get a lower bound of $\Omega(n^{1+\varepsilon})$ queries for better than an $n^{\delta}/4$ approximation. We can re-scale to get the same lower bound for an n^{δ} approximation, proving the theorem. Finally, we have the constraint $n^{2\varepsilon+3\delta}\log^3 n = o(n)$. ∎

10.3.3 Limitations of the Algorithm

High-Level Proof Plan. In this section, we show the limitations of the algorithm Alg. Since Alg is restricted to making only $q = n^{1+\varepsilon}$ queries, it can uncover only a small portion of the graph. Furthermore, because every vertex in the core has many edges to dummy vertices (D), the observed core subgraph—denoted E^C_{obs}—is extremely sparse. To be more precise, the main lemma of this section states that, with high probability, the subgraph observed by Alg in the core (E^C_{obs}) forms a disjoint union of stars. This structural property is key to showing that Alg cannot distinguish between the yes and no instances based on limited query access. We state the lemma formally below:

Lemma 10.3.5. *Let $G \sim \mathcal{D}$ and consider the algorithm* Alg *that makes $n^{1+\varepsilon}$ non-adaptive adjacency list queries. Then, with high probability, the edges* Alg *finds inside the core (E^C_{obs}) forms a disjoint union of stars.*

To establish this, we will prove that, with high probability, any discovered endpoint v of a discovered edge $u \to v$ neither discovers any additional edges in the core (Claim 10.3.8), nor has any other discovered edges incident on it (Claim 10.3.9). We will also establish some additional properties in this section that hold with high probability and will be used in the analysis of Section 10.4:

- The number of discovered edges in the core is small—at most $\widetilde{O}(n^{\varepsilon+\delta})$ (Claim 10.3.6).

Symbol / Term	Description
Happy vertex	A vertex that reveals at least one edge in the core through its queried indices.
S	Set of star centers which are also the set of happy vertices.
P_{YES}, P_{NO}	Set of petals (neighbors of star centers) in the yes and no cases, respectively.
τ-heavy vertex	A vertex with at least τ queries made on it. τ is taken as powers of 2: $\tau = 2^i$ for $i \in [\alpha, \beta]$.
α, β	$\alpha = \log(n^{\varepsilon} \log n)$ and $\beta = \log(16 n^{2\varepsilon+\delta} \log^2 n)$.

Table 10.3: Notation for Section 10.3.3.

- Any discovered endpoint v of a discovered edge $u \to v$ receives only a small number of queries, specifically $q_v \leqslant \widetilde{O}(n^{2\varepsilon+\delta})$ (Lemma 10.3.11). This lemma also shows that the number of petal vertices with many queries (below the previous threshold of $\widetilde{O}(n^{2\varepsilon+\delta})$) is small, revealing a tradeoff between the number of queries and the number of such vertices.
- Every vertex $v \in C$ has small in-degree from dummy vertices D, bounded by $O(n^{\varepsilon})$ (Claim 10.3.12).

To prove these claims, we rely on the randomness of the labeling permutation $\pi \in \Pi$ and the adjacency list $\ell \in \mathcal{L}$. The algorithm Alg specifies the number of queries for each vertex label such that they sum to $q = n^{1+\varepsilon}$. While the query distribution over labels in $\mathcal{U} \cup \mathcal{V}$ may be highly non-uniform, the random permutation π ensures that, in expectation, any fixed vertex (e.g., a_1) receives n^{ε} queries. For intuition, one can think of the analysis as if each vertex label receives n^{ε} queries; the bounds we derive will be the same even under the actual, potentially non-uniform query distribution.

The randomness used in the adjacency lists of core vertices comes from the random placement of core indices, determined by ψ and the random permutations σ_C, σ_D. For vertices in D, we use the randomness of the adjacency list being randomly permuted, as specified by σ. Importantly, we do not rely on the randomness of $\mathcal{D}^C_{\text{NO}}$ or $\mathcal{D}^C_{\text{YES}}$.

Finally, to prove the claims we need, we will set the following constraint:

$$n^{2\varepsilon+3\delta} \log^2 n = o(n). \tag{10.1}$$

We now turn to the formal claims and their proofs, as outlined in the high-level plan. We begin by showing that, with high probability, only a small number of core edges are discovered by Alg.

Claim 10.3.6. *With probability $1 - o(1)$, the number of core edges seen by the algorithm* Alg *is at most $4n^{\varepsilon+\delta} \log n$ in $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$.*

Proof. Consider any vertex v in C for which we make q_v queries. Since we pick a random labeling permutation π over the vertices, the probability that this vertex v is in A is $2n^{1-\delta}/(2n + 4n^{1-\delta}) \leqslant n^{-\delta}$ and the probability v is in B is at most 1.

Let the vertex v be in A. We want to calculate the expected number of core edges that are queried for v. Let $X_i^v = 1$ if the i^{th} core edge of v is queried for $i \in [n^\delta]$. Since the core indices are chosen randomly using $\psi(v)$, this just means that the i^{th} core edge lands in the queried region. We have that $\mathbb{E}[X_i^v] = \Pr(X_i^v = 1) = q_v/d^*$ since exactly q_v out of d^* indices are queried. $X^v = \sum_i X_i^v$ represents the number of core edges of v that land in the queried region. We have that $\mathbb{E}[X^v] = \sum_i \mathbb{E}[X_i^v] = n^\delta \cdot q_v/d^*$.

Let the vertex v be in B. We want to calculate the expected number of core edges that are queried for v. Since a B vertex has just one core edge this is the same as the probability that the core edge lands in the queried indices. Let $Y^v = 1$ if the core edge of v is queried for and 0 otherwise. We have that $\mathbb{E}[Y^v] = \Pr(Y^v = 1) = q_v/d^*$ since exactly q_v out of d^* indices are queried.

We now come back to the case where v is an A vertex or a B vertex with some probability. Let Z^v be the random variable that denotes the number of core edges of v that land in the queried region. We have:

$$\begin{aligned}\mathbb{E}[Z^v] &= \Pr(v \in A) \cdot \mathbb{E}[Z \mid v \in A] + \Pr(v \in B) \cdot \mathbb{E}[Z \mid v \in B] \\ &\leqslant n^{-\delta} \cdot n^\delta \cdot q_v/d^* + 1 \cdot q_v/d^* \\ &= 4n^\delta q_v/n.\end{aligned}$$

Let $Z = \sum_v Z^v$ be the total number of discovered edges in the core. The expected number of discovered edges is $\mathbb{E}[Z] = \sum_v \mathbb{E}[Z^v] \leqslant \sum_v 4n^\delta q_v/n = n^{1+\varepsilon} \cdot 4n^\delta/n = 4n^{\varepsilon+\delta}$. The second last equality is true because $\sum_v q_v = q = n^{1+\varepsilon}$. By Markov's inequality the probability that we exceed $\log n$ times the expectation is $1/\log n$, implying the bound in the claim. ∎

To prove Lemma 10.3.5, we define a vertex as **happy** if the queries on it reveal at least one edge in the core. We first calculate the probability that a vertex is happy.

Claim 10.3.7. *Let $a \in A$ be an arbitrary vertex and $v \in \mathcal{U} \cup \mathcal{V}$ be an arbitrary label. The probability that vertex a with label v is happy is at most $n^\delta \cdot q_v/d^*$.*

Proof. Since $v \in A$, it has n^δ core edges. Let $\mathcal{E}_i$ be the event that the i^{th} core edge lands in the queried portion for $i \in [n^\delta]$. We have $\Pr(\mathcal{E}_i) = q_v/d^*$ and $\Pr(v \text{ is happy}) = \Pr(\mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_{n^\delta}) \leqslant n^\delta \cdot q_v/d^*$ (union bound). ∎

We now show that the incoming endpoint of every discovered edge in the core is not happy.

Claim 10.3.8. *With high probability, the incoming endpoint of every discovered edge in the core is not a happy vertex, in both $\mathcal{D}^C_{YES}$ and $\mathcal{D}^C_{NO}$.*

Proof. Consider the argument for A-to-A edges. The cases of A-to-B edges and B-to-B edges can only have better bounds because the probability that a vertex in B is happy is $1/d^*$ which is much smaller than n^δ/d^*. Fix any A-to-A edge $e = (a^L_i, a^R_j)$. Since π is a random labeling permutation, a^L_i and a^R_j get different labels implying different queries to their adjacency lists depending on the label.

Consider the case where a^L_i gets label u and a^R_j gets label v. This happens with probability $1/n'^2$. We have $\Pr(u \text{ discovers } e) = q_u/d^*$ and $\Pr(v \text{ is happy}) \leqslant n^\delta \cdot q_v/d^*$ (Claim 10.3.7). Also, note that these events are independent of each other because they only depend on the randomness of $\psi(u), \psi(v)$ in their adjacency lists. We now calculate the probability that $e = (a^L_i, a^R_j)$ is happy:

$$\begin{aligned}
\Pr(e \text{ is happy}) &= \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} (1/n'^2) \cdot \Pr(u \text{ discovers } e) \cdot \Pr(v \text{ is happy}) \\
&\leqslant \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} (1/n^2) \cdot (q_u/d^*) \cdot (n^\delta q_v/d^*) \\
&\leqslant 4n^{3\delta}/n^4 \sum_{u \in \mathcal{U}} q_u \sum_{v \in \mathcal{V}} q_v \\
&\leqslant (4n^{3\delta}/n^4) \cdot n^{2+2\varepsilon} \\
&= 4n^{2\varepsilon+3\delta-2}.
\end{aligned}$$

The same calculation also works for A-to-B and B-to-B edges with better bounds however we just use the worst-case bound obtained here. Since there are $2n$ edges in the core, the expected number of happy edges is at most $2n \cdot 4n^{2\varepsilon+3\delta-2} = 8n^{2\varepsilon+3\delta-1}$. We have that $\log n$ times the expected value is $8n^{2\varepsilon+3\delta-1} \cdot \log n = o(1)$ (Eq (10.1)). Thus, by Markov's inequality, we conclude that no edge is happy with probability $1 - o(1)$. ∎

We also show that no pair of vertices share a discovered neighbor with high probability.

Claim 10.3.9. *With high probability, no pair of vertices in the core discover an edge to the same neighbor for $\mathcal{D}^C_{YES}$ and $\mathcal{D}^C_{NO}$.*

Proof. No pair of A, B vertices can share a neighbor in either case. B vertices cannot share a neighbor in the yes case and A vertices cannot share a neighbor in the no case. Consider the argument for A vertices in the yes case. The exact same argument works for B vertices in the no case.

We fix any two vertices $a, a' \in A$ that share a neighbor a''. The number of possible choices for a are $n^{1-\delta}$. After fixing a, a' has $n^{2\delta}$ choices because a has n^δ neighbors each of which can have n^δ different neighbors. Thus, we fix a pair a, a' that share neighbor a'' out of the $n^{1+\delta}$ choices and calculate the probability that both edges $e = (a, a'')$ and $e' = (a', a'')$ are discovered.

Consider the case where a gets label v and a' gets label v' ($v' \neq v$). This happens with probability $1/n'(n'-1)$. This is slightly different from the previous proof because both the vertices are on the same side. $\Pr(e \text{ is discovered}) \leqslant q_v/d^*$ and $\Pr(e' \text{ is discovered}) = q_{v'}/d^*$. We now calculate the probability that both e, e' are discovered:

$$\begin{aligned}
\Pr(a, a' \text{ are happy}) &= \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V} - \{v\}} (1/n'(n'-1)) \cdot \Pr(v \text{ and } v' \text{ are happy}) \\
&\leqslant \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} (1/n^2) \cdot (q_v/d^*) \cdot (q_{v'}/d^*) \\
&\leqslant 4n^{2\delta}/n^4 \sum_{v \in \mathcal{V}} q_v \sum_{v' \in \mathcal{V}} q_{v'} \\
&\leqslant (4n^{2\delta}/n^4) \cdot n^{2+2\varepsilon} \\
&= 4n^{2\varepsilon+2\delta-2}.
\end{aligned}$$

Since there are $n^{1+\delta}$ choices for a, a', the expected number of pairs of vertices in A that share a discovered neighbor is at most $n^{1+\delta} \cdot 4n^{2\varepsilon+2\delta-2} = 4n^{2\varepsilon+3\delta-1}$. We have that $\log n$ times the expected value is $4n^{2\varepsilon+3\delta-1} \cdot \log n = o(1)$ (by Eq (10.1)). Thus, by Markov's inequality, we conclude that no pair of happy vertices in A share a neighbor with probability $1 - o(1)$.

We now consider B vertices sharing an edge in the no case. We fix any two vertices $b, b' \in B$ that share a neighbor. The number of possible choices for b are n. After fixing b, b' has n^δ choices because b has 1 neighbor which has n^δ different neighbors. Thus, we fix a pair b, b' out of the $n^{1+\delta}$ choices and calculate the probability that both of them are happy.

Consider the case where b gets label v and b' gets label v' ($v' \neq v$). This happens with probability $1/n'(n'-1)$. v is happy with probability q_v/d^* since it has just 1 core edge.

Similarly, v' is happy with probability $q_{v'}/d^*$. We now calculate the probability that both b, b' are happy:

$$\begin{aligned}
\Pr(b, b' \text{ are happy}) &= \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V} - \{v\}} (1/n'(n'-1)) \cdot \Pr(v \text{ and } v' \text{ are happy}) \\
&\leqslant \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} (1/n^2) \cdot (q_v/d^*) \cdot (q_{v'}/d^*) \\
&\leqslant 4n^{2\delta}/n^4 \sum_{v \in \mathcal{V}} q_v \sum_{v' \in \mathcal{V}} q_{v'} \\
&\leqslant (4n^{2\delta}/n^4) \cdot n^{2+2\varepsilon} \\
&= 4n^{2\varepsilon+2\delta-2}.
\end{aligned}$$

Since there are $n^{1+\delta}$ choices for b, b', the expected number of pairs of happy vertices in B that share a neighbor is at most $n^{1+\delta} \cdot 4n^{2\varepsilon+2\delta-2} = 4n^{2\varepsilon+3\delta-1}$. We have that $\log n$ times the expected value is $4n^{2\varepsilon+3\delta-1} \cdot \log n = o(1)$ (by Eq (10.1)). Thus, by Markov's inequality, we conclude that no pair of happy vertices in B share a neighbor with probability $1 - o(1)$. We can union bound over both cases and conclude the proof. ∎

Having established the necessary preliminaries, we now proceed to prove Lemma 10.3.5.

Proof of Lemma 10.3.5. We condition on the high probability events in Claim 10.3.8 and Claim 10.3.9. Consider a happy vertex v and its discovered edge $e = v \to u$. We know from Claim 10.3.8 that the discovered endpoint u of a discovered edge cannot be happy. Thus, there are no discovered edges going out of u. Also, we know by Claim 10.3.9 that no pair of vertices in the core discover an edge to the same vertex. Thus, there are no other discovered edges incident on u. Therefore, the edges Alg finds inside the core (E^C_{obs}) forms a disjoint union of stars with high probability. ∎

The answers to the queries made by Alg form a disjoint union of stars in the subgraph induced by the core. The center of each star can be either an A or B vertex. We let S denote the set of **star centers** and we call their discovered neighbors **petals** denoted by P_{YES} in the yes case and P_{NO} in the no case. Note that the center of a star is always a happy vertex i.e. the vertex that was queried, resulting in an edge within the core being discovered. If the center is A then the star has degree at most n^δ. Otherwise, if the center is B then the star has degree 1. Alg can figure out the labels of the centers of the stars based on their degree. The important part is that all the neighbor queries made on the petals of the stars are D vertices, so we do not know for sure whether they are A or B

vertices. However, if there are many queries made on a petal vertex then Alg can figure out with some advantage whether the petal vertex comes from A or B. We now show that the number of queries on a petal vertex is actually small.

Let a vertex be τ-**heavy** if the number of queries made on it are at least τ. Given that the total number of queries Alg makes is $q = n^{1+\varepsilon}$, there are at most $n^{1+\varepsilon}/\tau$ τ-heavy vertices. We now show that the number of queries made on petal vertices are less than τ. For this to be false an edge must be discovered and its petal endpoint must be heavy. We show the number of such edges is 0 with high probability.

Claim 10.3.10. *The number of edges discovered with τ-heavy petal vertices is at most $8n^{2\varepsilon+\delta}\log^2 n/\tau$ with probability $1 - 2/\log^2 n$.*

Proof. Consider the argument for A-to-A edges. The proof is identical for other types of edges. Fix an A-to-A edge $e = (a_i^L, a_j^R)$. Since π is a random labeling permutation, a_i^L and a_j^R get random labels implying a different number of queries to their adjacency lists depending on the label. There are at most $n^{1+\varepsilon}/\tau$ τ-heavy vertex labels. Thus, the probability that a_j^R gets a heavy label is $n^{1+\varepsilon}/\tau \cdot n'$.

Consider the case where a_i^L gets label u. This happens with probability $1/n'$. $\Pr(u \text{ discovers } e) = q_u/d^*$. Note that the set of labels $\mathcal{U}$ and $\mathcal{V}$ are disjoint so the event that u discovers e is independent of the event that a_j^R picks a τ-heavy label. We now calculate the probability that $e = (a_i^L, a_j^R)$ is discovered-heavy:

$$\begin{aligned}
\Pr(e \text{ is discovered-heavy}) &= \sum_{u\in\mathcal{U}} (1/n') \cdot \Pr(u \text{ discovers } e) \cdot \Pr(v \text{ is } \tau\text{-heavy}) \\
&\leqslant \sum_{u\in\mathcal{U}} (1/n) \cdot (q_u/d^*) \cdot (n^{1+\varepsilon}/\tau \cdot n) \\
&= (2n^{\delta-2}) \cdot (n^{\varepsilon}/\tau) \cdot n^{1+\varepsilon} \\
&= 2n^{2\varepsilon+\delta-1}/\tau.
\end{aligned}$$

Since there are $2n$ edges in the core, the expected number of discovered-heavy edges is at most $4n^{2\varepsilon+\delta}/\tau$. Thus, by Markov's inequality, the probability that the number of discovered-heavy edges is $\log^2 n$ times the expectation is $1/\log^2 n$. Note that the argument is the same for heavy-discovered and discovered-heavy edges, so we get at most $8n^{2\varepsilon+\delta}\log^2 n/\tau$ such edges with probability $1 - 2/\log^2 n$ by a union bound. ∎

This claim implies that with high probability, there are no τ-heavy petals with $\tau = 16n^{2\varepsilon+\delta}\log^2 n$. We have to be a bit careful with the other end. If we set $\tau = 1$, we

get that there are at most $8n^{2\varepsilon+\delta}$ τ-heavy petals with high probability. But we know by Claim 10.3.6 that the number of discovered edges which is same as the number of petals is at most $4n^{\varepsilon+\delta}$. The reason we get an extra factor of n^{ε} is that we count the number of τ-heavy vertices as $n^{1+\varepsilon}/\tau$. But when $\tau = 1$ we have a tighter bound of $O(n)$ because there are only those many vertices in total. Thus, at the lower end, we will set $\tau = n^{\varepsilon}$, giving us at most $8n^{\varepsilon+\delta}\log^2 n$ with high probability. We now give the following lemma that captures these extreme cases along with others in the middle.

Lemma 10.3.11. *For all $i \in [\alpha, \beta]$, with high probability, the number of edges discovered with 2^i-heavy petal vertices is at most $2^{-i} \cdot 8n^{2\varepsilon+\delta}\log^2 n$ for $\alpha = \log(n^{\varepsilon}\log n)$ and $\beta = \log(16n^{2\varepsilon+\delta}\log^2 n)$.*

Proof. We condition on the event in Claim 10.3.10 for $\tau = 2^i$, for all $i \in [\alpha, \beta]$. To account for all such values of τ, we apply a union bound over the failure probabilities of all the cases. Since the number of cases are at most $\beta \leqslant \log n$ (by Eq (10.1)), the total failure probability is at most $\log n \cdot (2/\log^2 n) = o(1)$. ∎

Finally, we show that a vertex does not have many discovered edges from D. Any fixed vertex v in the core has in-degree at most $4n^{\varepsilon}$ from D.

Claim 10.3.12. *Every vertex $v \in A \cup B$ has in-degree at most $4n^{\varepsilon}$ from D in the observed graph with high probability.*

Proof. Fix a vertex $v \in C$. We will calculate the number of discovered edges from D to v. v has at most d^* neighbors to uniformly random vertices in D. We let X_i be a random variable that is 1 if the i^{th} edge of v to D is discovered and 0 otherwise for $i \in [d^*]$. We are interested in $X = \sum_{i\in[d^*]} X_i$ which is the total number of edges of v to D that are discovered.

We now focus on the random variable X_i. Let the i^{th} edge of v to D fall on a vertex d. Since the labeling permutation is random, d gets assigned a random label. Assume the label is u which has q_u queries. This happens with probability $1/n'$. The probability that the edge is discovered when the label is u is $q_u/\deg(u)$. Thus, the probability that $X_i = 1$ is:

$$\begin{aligned}\Pr(X_i = 1) &= (1/n')\sum_{u\in\mathcal{U}} q_u/\deg(u) \\ &\leqslant (4/n^2)\sum_{u\in\mathcal{U}} q_u\end{aligned}$$

$$= 4n^{\varepsilon-1}.$$

This implies that $\mathbb{E}[X] = \sum_{i\in[d^*]} \mathbb{E}[X_i] = \sum_{i\in[d^*]} \Pr(X_i = 1) \leqslant d^* \cdot 4n^{\varepsilon-1} = 2n^{\varepsilon-\delta} \leqslant 2n^{\varepsilon} = \mu_H$. Observe that the X_i's are independent because each edge to D is an independent sample with replacement. Also, each X_i is in $[0,1]$ so we can apply the Chernoff bound (Proposition 2.4.3) and get:

$$\Pr(X > 2\mu_H) \leqslant \exp(-\mu_H/4) \ll 1/\text{poly}(n).$$

Therefore, the in-degree of v from D is at most $4n^{\varepsilon}$ with probability $1 - 1/\text{poly}(n)$. We can union bound this over vertices $v \in C$ and get that with high probability, all the vertices have in-degree at most $4n^{\varepsilon}$ from D. ∎

For the rest of the proof we condition on the high probability events in Claims 10.3.2, 10.3.6 and 10.3.12 and Lemmas 10.3.5 and 10.3.11.

10.4 The Coupling Argument

High Level Idea. In this section, we will use coupling to show that after conditioning on some high probability events, the probability that an observed graph comes from $\mathcal{D}_{\text{NO}}$ is roughly $1/2$ and the probability it comes from $\mathcal{D}_{\text{YES}}$ is roughly $1/2$. Note that the support of $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ are completely disjoint. However, the algorithm Alg only makes a few queries on this graph revealing a small portion of the graph. For most choices of the revealed part of the graph called the observed graph, it looks like it could have come from $\mathcal{D}_{\text{NO}}$ or $\mathcal{D}_{\text{YES}}$ with roughly equal probability.

Ideally we want the coupling to be a perfect matching between elements of $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ such that when we uniformly sample an element of $\mathcal{D} = \frac{1}{2}\mathcal{D}_{\text{YES}} + \frac{1}{2}\mathcal{D}_{\text{NO}}$ we can do the sampling by picking a uniformly random edge of the matching between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ and then randomly choosing an endpoint (we will have $|\mathcal{D}_{\text{NO}}| = |\mathcal{D}_{\text{YES}}|$). The property the matching has is that for any edge in the matching the observed graph is exactly the same for both the endpoints of the edge. This means that any algorithm cannot tell apart the endpoints of an edge with probability better than $1/2$. This sampling process is the same as sampling a uniformly random element from the no case and the yes case independently and then picking one of them with probability $1/2$. The problem with this however is that the samples in the no and yes case could give different observed graphs. We crucially need the observed graphs to be the same, so we can argue that the algorithm Alg does not know whether the instance that the observed graph came from is a yes instance or a no instance. This is the reason we need a coupling

Symbol / Term	Description
$\mathcal{H} = (\mathcal{X} \cup \mathcal{Y}, \mathcal{M})$	Bipartite graph used to define the coupling between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$.
$\mathcal{X}, \mathcal{Y}$	Sets of nodes in $\mathcal{H}$, where each node is a tuple consisting of a graph instance from $\mathcal{D}_{\text{NO}}$ or $\mathcal{D}_{\text{YES}}$, a labeling permutation $\pi \in \Pi$, and a set of adjacency lists $\ell \in \mathcal{L}$.
$\mathcal{M}$	A matching between $\mathcal{X}$ and $\mathcal{Y}$ of size $(1 - o(1)) \cdot \lvert\mathcal{X}\rvert$, where each edge connects nodes whose observed graphs are identical.
block	A subset of nodes in $\mathcal{X}$ or $\mathcal{Y}$ defined by fixing part of the underlying randomness (graph, permutation, or adjacency lists).
bad node	A node that violates one or more of the high-probability events we condition on.
block-compliant	A property of matching $\mathcal{M}$ such that, when blocks are treated as supernodes, $\mathcal{M}$ remains a valid matching on the resulting supernode graph.
confined matching	The portion of matching $\mathcal{M}$ restricted to operate within the current partition of blocks, preserving block-compliance.

Table 10.4: Notation for Section 10.4.

Unfortunately, we will not get a perfect matching between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ such that the observed graphs for the endpoints of each matching edge are identical. However, if we ignore a $o(1)$ fraction of elements, then we indeed get a perfect matching between the two distributions (call this the successful matching). We will add an arbitrary perfect matching between the ignored $o(1)$ fraction of elements and call this the failed matching. When $\mathcal{D}$ samples an edge of the failed matching the algorithm Alg succeeds with probability 1 in distinguishing between the two cases. When $\mathcal{D}$ samples an edge of the successful matching the algorithm Alg succeeds with probability $1/2$ in distinguishing between the two cases. Since the fraction of edges in the failed matching is $o(1)$ and in the successful matching is $(1 - o(1))$, Alg is correct only with probability $1/2 + o(1) < 2/3$. Finally, we will need the following constraint in this section:

$$n^{2\varepsilon+3\delta} \log^3 n = o(n). \tag{10.2}$$

We will fix elements in $\mathcal{D}_{\text{NO}}$ and then show which element in $\mathcal{D}_{\text{YES}}$ they are coupled with. When constructing the coupling we first couple the edges in the core which is a coupling between distributions $\mathcal{D}^C_{\text{NO}}$ with $\mathcal{D}^C_{\text{YES}}$. We then couple the labeling permutations which come from Π which is the set of all labeling permutations. Then we couple the edges to D which come from the distribution $\mathcal{D}^D$ which is the same in the no and the yes case. We finally couple the adjacency lists between the two cases which come from $\mathcal{L}$. This is just a rough idea of how the coupling works on a high level—the final picture is more intricate. To be very clear about the coupling, we define the following bipartite graph $\mathcal{H} = (\mathcal{X} \cup \mathcal{Y}, \mathcal{M})$.

Each node of the graph in the no case (set $\mathcal{X}$) corresponds to a graph G_{NO} from $\mathcal{D}_{\text{NO}}$,

a labeling permutation π from Π and the adjacency lists of all the vertices denoted by $\ell \in \mathcal{L}$. The nodes of the graph in the yes case (set $\mathcal{Y}$) correspond to a graph G_{YES} from $\mathcal{D}_{\text{YES}}$, a permutation π from Π and permutations of the adjacency lists of all the vertices denoted by $\ell \in \mathcal{L}$. Recall that the random variable $\ell(v)$ for adjacency lists of vertices $v \in C$ corresponds to picking the locations of the core indices uniformly at random using $\psi(v)$, randomly permuting the core neighbors using $\sigma_C(v)$ and the dummy neighbors using $\sigma_D(v)$. The random variable $\ell(v)$ for adjacency lists of vertices $v \in D$ corresponds to v's edges to the core and randomly permuting the entire adjacency list using $\sigma(v)$. Note that we will use the term nodes for elements in $\mathcal{X}$ and $\mathcal{Y}$, and reserve the term vertices for the graphs in $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$.

We will add edges to this graph and ensure that the edges of the graph $\mathcal{M}$ forms a matching. We will also have the property that for any edge $(x, y) \in \mathcal{M}$, the observed graph on x and y will be identical. We first want to show that the sizes of $\mathcal{X}$ and $\mathcal{Y}$ are the same. We start by showing that the number of possible core graphs in the no case ($\mathcal{D}^C_{\text{NO}}$) and the yes case ($\mathcal{D}^C_{\text{YES}}$) are the same.

Claim 10.4.1. $\left|\mathcal{D}^C_{NO}\right| = \left|\mathcal{D}^C_{YES}\right|$ *and* $|\mathcal{D}_{NO}| = |\mathcal{D}_{YES}|$.

Proof. In $\mathcal{D}^C_{\text{NO}}$, the randomness comes from the matchings between A^R and B^L. There is a random perfect matching between A^R and each group of B^L. There are n^δ groups each of size n/n^δ. In $\mathcal{D}^C_{\text{YES}}$, the randomness comes from the matchings between A^L and A^R. There are n^δ random perfect matchings each of size n/n^δ.

Thus, both cases have n^δ random perfect matchings of size n/n^δ implying that the number of possible core graphs are the same in both cases ($\left|\mathcal{D}^C_{\text{NO}}\right| = \left|\mathcal{D}^C_{\text{YES}}\right|$). This also implies that $|\mathcal{D}_{\text{NO}}| = |\mathcal{D}_{\text{YES}}|$ because $\mathcal{D}_{\text{NO}}$ is a product distribution of $\mathcal{D}^C_{\text{NO}}$ and $\mathcal{D}^D$, and $\mathcal{D}_{\text{YES}}$ is a product distribution of $\mathcal{D}^C_{\text{YES}}$ and $\mathcal{D}^D$. ∎

We use this to prove that the support sizes of $\mathcal{X}$ and $\mathcal{Y}$ are the same.

Claim 10.4.2. $|\mathcal{X}| = |\mathcal{Y}|$.

Proof. Claim 10.4.1 proves that $\left|\mathcal{D}^C_{\text{NO}}\right| = \left|\mathcal{D}^C_{\text{YES}}\right|$. The distribution $\mathcal{D}^D$ is used in both $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$ and is independent of $\mathcal{D}^C_{\text{NO}}$ and $\mathcal{D}^C_{\text{YES}}$, so we pick an arbitrary fixing of $\mathcal{D}^D$. We now consider the set of labeling permutations Π which is the same for both cases and independent of $\mathcal{D}_{\text{NO}}, \mathcal{D}_{\text{YES}}$. Thus, we pick an arbitrary fixing of Π.

We finally look at the different possibilities for the adjacency lists in $\mathcal{L}$. For every vertex in the core $v \in C$, $\psi(v), \sigma_C(v)$ and $\sigma_D(v)$ have the same support size in the no and the yes

case because we just have to randomly choose the core indices from the d^* total indices and then randomly permute both parts. For every dummy vertex $v \in D$, the number of adjacency lists is the same in both cases since they are randomly permuted ($\deg(v)!$ possibilities in both cases).

For each fixing of $\mathcal{D}^D$ and Π we have that $|\mathcal{L}|$ is the same in both cases and $\left|\mathcal{D}^C_{\text{NO}}\right| = \left|\mathcal{D}^C_{\text{YES}}\right|$. Therefore, we can conclude that $|\mathcal{X}| = |\mathcal{Y}|$. ∎

We now want to show that the matching $\mathcal{M}$ is large. We will prove that $\mathcal{M}$ matches $(1 - o(1))$ fraction of the vertices in $\mathcal{X}$.

Lemma 10.4.3. $|\mathcal{M}| \geqslant (1 - o(1))\,|\mathcal{X}|$.

We will use this to prove Lemma 10.3.4.

Proof of Lemma 10.3.4. Consider algorithm Alg that makes $q = n^{1+\varepsilon}$ queries and distinguishes between $\mathcal{D}_{\text{NO}}$ and $\mathcal{D}_{\text{YES}}$. For any edge $e \in \mathcal{M}$, the observed graph for both the endpoints is the same (property of the edges of $\mathcal{M}$). So any algorithm cannot do better than guessing when asked to distinguish between the endpoints of $\mathcal{M}$ and will be wrong with probability 1/2.

We will add an arbitrary perfect matching between the ignored $o(1)$ fraction of elements and call this the **failed** matching $\mathcal{M}'$. For any edge $e \in \mathcal{M}'$, Alg can be correct with probability at most 1. Also, note that $\mathcal{M}^* = \mathcal{M} \cup \mathcal{M}'$ is a perfect matching between $\mathcal{X}$ and $\mathcal{Y}$.

We now look at the distribution $\mathcal{D} = \frac{1}{2}\mathcal{D}_{\text{YES}} + \frac{1}{2}\mathcal{D}_{\text{NO}}$ that samples a random edge of $\mathcal{M}^*$ and then samples a random endpoint of that edge which is a random sample from $\mathcal{D}$. When $\mathcal{D}$ samples an edge of the successful matching $\mathcal{M}$, the algorithm Alg succeeds with probability 1/2 in distinguishing between the two cases. When $\mathcal{D}$ samples an edge of the failed matching $\mathcal{M}'$, the algorithm Alg succeeds with probability at most 1 in distinguishing between the two cases. Finally, the number of edges in the failed matching is $o(1) \cdot |\mathcal{X}|$ and in the successful matching is $(1 - o(1)) \cdot |\mathcal{X}|$. Thus, Alg is correct only with probability at most $(1 - o(1)) \cdot (1/2) + o(1) \cdot 1 \leqslant 1/2 + o(1)$.

We finally note that we have constraints Eq (10.1) and Eq (10.2) but Eq (10.1) is weaker giving us the final constraint: $n^{2\varepsilon+3\delta} \log^3 n = o(n)$. ∎

We will now prove Lemma 10.4.3 in the rest of the section. The idea is to divide $\mathcal{X}$ and $\mathcal{Y}$ into blocks such that each block corresponds to a potentially different observed

graph. However, all the nodes in the block crucially have the same observed graph. We will create these blocks by **fixing** some part of the randomness of the distributions. Recall that $\mathcal{X}$ and $\mathcal{Y}$ are the supports of distributions over tuples consisting of a graph, a labeling permutation, and a set of adjacency lists. By fixing parts of the randomness within these distributions, we can partition $\mathcal{X}$ and $\mathcal{Y}$ into blocks, where each block corresponds to a particular fixing of that randomness. For instance, if we fix the first element of the tuple to take on different graphs from $\mathcal{D}_{\text{NO}}$, then $\mathcal{X}$ is partitioned into $|\mathcal{D}_{\text{NO}}|$ blocks, each corresponding to a particular graph from $\mathcal{D}_{\text{NO}}$.

We will have the property that if we think of the blocks as supernodes, $\mathcal{M}$ still forms a matching on the supernode graph. This introduces the notion of the matching being **block-compliant**. The matching $\mathcal{M}$ is block-compliant with respect to some fixing if for every edge $e \in \mathcal{M}$ the following holds. Say edge e is between two blocks $\mathcal{X}_1$ and $\mathcal{Y}_1$ then any edge incident on a vertex in $\mathcal{X}_1$ goes to $\mathcal{Y}_1$ and vice versa.

We will then show that if we match two blocks (on the supernode level) then their sizes are within a $(1 \pm o(1))$ factor of each other so $\mathcal{M}$ matches all the nodes in one block and most nodes in the other. We ignore a $o(1)$ fraction of nodes overall because there is a $o(1)$ fraction of nodes that does not satisfy the high probability events we condition on. Also, when we match two blocks, their sizes could be a $(1 \pm o(1))$ factor apart. Therefore, we get that matching $\mathcal{M}$ has size $(1 - o(1)) \cdot (1 - o(1)) \cdot |X| = (1 - o(1)) \cdot |X|$.

We call a node **bad** if it does not satisfy the high probability events we condition on and **good** otherwise. We will call a block is **bad** if all the nodes in it are bad. We will show that every bad node will be part of a bad block, and we will ignore bad blocks. These bad blocks we ignore will only be a $o(1)$ fraction of all the nodes because they are filled only with bad nodes and the bad nodes only form $o(1)$ fraction of all the nodes. We will also say that a block is **good** if it is not bad. Since all bad nodes belong to bad blocks, a good block will only contain good nodes.

We will not fix the randomness in one shot and get the blocks we want. We will start with blocks of a large size and then divide them into finer and finer blocks. Each time we fix some part of the randomness we will get sub-blocks of a block. We will find bad blocks at almost every level of this partitioning, and we will ignore such blocks. We will consider arbitrary but fixed fixings on both sides that give good blocks and will keep $\mathcal{M}$ **confined** within those blocks. This means that the matching $\mathcal{M}$ will be block-compliant at each level of the partitioning into finer blocks and will have edges incident on only good blocks (see Figure 10.3).

10.4.1 Coupling the Core

Here we describe the coupling for the core of the graph. We consider an arbitrary core G^C_{NO} in the no case $\mathcal{D}^C_{\text{NO}}$. First consider the edges $(b^L_{i,j}, a^R_{j'})$ and $(b^R_{i,j}, a^L_j)$ we have in G^C_{NO} for all $i \in [n^\delta]$ and $j \in [n/n^\delta]$. We will remove these edges and add edges $(b^L_{i,j}, b^R_{i,j})$ and $(a^L_j, a^R_{j'})$ which belong to G^C_{YES}. These four edges together form a **quad** in the coupling.

Using this we can also define the notion of **corresponding** edges. For any edge (u, v) in G^C_{NO} directed out of u, its corresponding edge in G^C_{YES} is the only other edge in the coupling **quad** incident on u and directed out of u. We can similarly define this for (u, v) in G^C_{YES} as well. For instance, consider any edge $(b^L_{i,j}, a^R_{j'})$ in the core in G^C_{NO} and consider the direction $b^L_{i,j} \to a^R_{j'}$. Its corresponding edge in the yes case will be the edge $b^L_{i,j} \to b^R_{i,j}$. On the other hand, if we had considered the direction $a^R_{j'} \to b^L_{i,j}$ then the corresponding edge in the yes case would have been the edge $a^R_{j'} \to a^L_j$.

This procedure describes the coupling for all the edges within the core. The instances shown in Figure 10.2 illustrate examples of such coupled cores. We now show that this coupling, denoted by $\mathcal{M}^C$, always gives us an instance of $\mathcal{D}^C_{\text{YES}}$.

Claim 10.4.4. *If we start with any $G^C_{NO} \in \mathcal{D}^C_{NO}$ and apply the coupling procedure we get a core graph $G^C_{YES} \in \mathcal{D}^C_{YES}$.*

Proof. This is because firstly we get edges (b^L_i, b^R_i) for all $i \in [n]$. Secondly, every group B^L_i for $i \in [n^\delta]$ gives a uniformly random matching between A^L and A^R. This is because the matching between B^L and B^R is an identity matching in the yes case and the matching between B^R and A^L is an identity matching in the no case. So if there is an edge between $b^L_{i,j}$ and $a^R_{j'}$ in the no case then there is an edge between a^L_j and $a^R_{j'}$ in the yes case. Therefore, the matching between B^L_i and A^R in the no case is the same as the matching between A^L and A^R in the yes case if we relabel $b^L_{i,j}$ with a^L_j. Thus, we get n^δ uniformly random matchings between A^L and A^R in the yes case concluding the proof. ∎

We now show that this coupling indeed gives a perfect matching.

Claim 10.4.5. *The coupling given by $\mathcal{M}^C$ is a perfect matching between $\mathcal{D}^C_{NO}$ and $\mathcal{D}^C_{YES}$.*

Proof. To show that this coupling is a perfect matching we need to show that the sizes of both sides are equal and that every node has degree exactly 1 in this coupling. Note that Claim 10.4.1 showed that $\left|\mathcal{D}^C_{\text{NO}}\right| = \left|\mathcal{D}^C_{\text{YES}}\right|$. Also, note that the degree of a node in $\mathcal{D}^C_{\text{NO}}$ is exactly 1 because the coupling gives exactly one instance of $\mathcal{D}^C_{\text{YES}}$ (Claim 10.4.4). All we need to show now is that every node in $\mathcal{D}^C_{\text{YES}}$ has degree 1 in this coupling.

Symbol / Term	Description
$G^C_{\text{NO}}, G^C_{\text{YES}}$	Core subgraphs of G_{NO} and G_{YES}, sampled from $\mathcal{D}^C_{\text{NO}}$ and $\mathcal{D}^C_{\text{YES}}$, respectively.
$\mathcal{M}^C$	Coupling (perfect matching) between $\mathcal{D}^C_{\text{NO}}$ and $\mathcal{D}^C_{\text{YES}}$ that ensures the observed core subgraphs are identical for matched endpoints.
quad	A group of four edges (two from G^C_{NO}, two from G^C_{YES}) that define how individual edges are coupled.
corresponding edge	For an edge $u \to v$ in G^C_{NO}, the coupled edge $u \to v'$ in G^C_{YES} belonging to the same quad (or vice versa).
$\mathcal{X}_1, \mathcal{Y}_1$	Subsets of $\mathcal{X}, \mathcal{Y}$ corresponding to a fixed core structure G^C_{NO} and its coupled core structure G^C_{YES}

Table 10.5: Notation introduced in Section 10.4.1.

Consider any $G^C_{\text{YES}} \in \mathcal{D}^C_{\text{YES}}$ which is defined by matchings $M_1, M_2, \ldots, M_{n^\delta}$. We will now invert the coupling to show exactly one pre-image. Consider any matching M_i for $i \in [n^\delta]$ and groups B^L_i, B^R_i in B. We know that B^R_i and A^L have an identity matching between them in the no case and B^L_i and B^R_i have an identity matching between them in the yes case.

Consider an arbitrary edge between $(a^L_j, a^R_{j'})$ in the yes case. We look at the quad this edge is a part of in the coupling. This includes edges $(b^L_{i,j}, b^R_{i,j})$ and $(a^L_j, b^R_{i,j})$. To get the edge $(a^L_j, a^R_{j'})$ using the coupling, the last edge in the quad has to be $(b^L_{i,j}, a^R_{j'})$. If we had some other edge $(b^L_{i,j}, a^R_{j''})$ in the no case, then we would get the edge $(a^L_j, a^R_{j''})$ in the yes case.

Thus, every node in the yes case has a unique pre-image in the coupling implying that the coupling is a perfect matching between $\mathcal{D}^C_{\text{NO}}$ and $\mathcal{D}^C_{\text{YES}}$. The matching between B^L_i and A^R in the no case is essentially the same matching as the one between A^L and A^R in the yes case (up to changing the names of the vertices in a fixed order). ∎

Therefore, we can fix any arbitrary structure for the core G^C_{NO} in the no case and its coupled structure G^C_{YES} in the yes case. This fixing creates blocks, and we need to ensure that $\mathcal{M}$ is block-compliant. We define the set of nodes $\mathcal{X}_1 \subseteq \mathcal{X}$ as $\mathcal{X}$ restricted to G^C_{NO} i.e. the nodes where the structure of the core is G^C_{NO}. Similarly, define $\mathcal{Y}_1 \subseteq \mathcal{Y}$ as $\mathcal{Y}$ restricted to G^C_{YES}. Finally, we note that the sizes of these sets of nodes will be the same under any fixing because everything else (labeling permutation, edges to D, permutation of the adjacency lists) is independent of it. For any fixing of the core $(\mathcal{X}_1, \mathcal{Y}_1)$, we will ensure that the matching $\mathcal{M}$ is confined to it (see Figure 10.3). This also makes sense because if we go to a block with a different structure then the observed graph could be different but

for every edge in $\mathcal{M}$ the endpoints should have the same observed graph. For the rest of the proof, we will only look at the induced subgraph on $\mathcal{X}_1$ and $\mathcal{Y}_1$.

Fixings So Far

- Section 10.4.1: Fixed structure for the core: G^C_{NO} in the no case and its coupled structure G^C_{YES} in the yes case.

10.4.2 Fixing the Permutation and the Star Centers

We now fix an arbitrary labeling permutation $\pi \in \Pi$ in both $\mathcal{X}_1$ and $\mathcal{Y}_1$. We also fix the set S of all the star centers (these are the set of vertices in the core that are happy). Lastly, we also fix $\psi(S), \sigma_C(S)$ which fixes the locations of the core indices in the adjacency lists of vertices in S and also the permutation of these core edges. Doing this fixes the entire observed graph in the core E^C_{obs} because the queries are deterministically fixed. We also note that there can be a bad choice of the fixing which disagrees with the high probability event in Lemma 10.3.5 (E^C_{obs} is not a disjoint union of stars) in the no or the yes case. The fixing is also bad if it disagrees with the high probability event in Claim 10.3.6 (the number of discovered edges are bounded) or Lemma 10.3.11 (the number of discovered-heavy edges are not bounded, which will happen if petals have many queries). In these cases, the entire block on at least one side will be bad, and so we can ignore this fixing.

Consider a star center v and its discovered neighbor u in the no case and its discovered neighbor u' in the yes case. Alg will see the edge (v, u) in the no case and the edge (v, u') in the yes case. Thus, the observed graphs will be different in both cases. This implies that $\mathcal{M}$ will not be confined to this particular fixing if we try to match the nodes in these blocks. To solve this problem, we construct a new permutation π' which is called the **dual permutation** of π. A **dual** permutation π' of a labeling permutation π is one which starts with π but swaps the label of every petal discovered in the no case (P_{NO}) with the label of the **corresponding petal** discovered in the yes case (P_{YES}). Consider a star center v and its petal $u \in P_{\text{NO}}$ in the no case. The corresponding edge for $v \to u$ is the edge $v \to u'$. Thus, $u' \in P_{\text{YES}}$ is the corresponding petal of u. We can similarly go from the yes case to the no case.

Note that π' is the same as π on all vertices in D (dummy vertices), all vertices in S (star centers), and all isolated vertices in E^C_{obs} (the observed graph inside the core). It differs only for vertices in $P := P_{\text{NO}} \cup P_{\text{YES}}$ under permutation π. Note that the sets P_{YES} and P_{NO} are different under π but P_{NO} under π will be the same as P_{YES} under π'. We define

P_{pair} to be the set of **petal pairs** where each petal pair is a set ($(u, u') \in P_{\text{pair}}$) containing a petal discovered in the no case ($u \in P_{\text{NO}}$) and its corresponding petal discovered in the yes case ($u' \in P_{\text{YES}}$). In π' we swap the labels within each petal pair in P_{pair}. Note that this is possible because E^C_{obs} is a disjoint union of stars, so discovered vertices in the no case and the yes case do not overlap for different vertices in S.

π' is called a dual permutation of π because if we start with permutation π' then its dual permutation will be π since we will swap back the labels.

Claim 10.4.6. *If we start with a labeling permutation π and get π' as the dual permutation then if we start with π' we will get π as the dual permutation.*

Proof. Note that we start with the labeling permutation π. The set S of star centers is fixed along with their core indices ($\psi(S)$). We then swap the names of the discovered neighbors (petals) of vertices in S in the no and the yes case. Notice however that when we start with the labeling permutation π', the queries on vertices in S will be the same and $\psi(S)$ is the same, so we will find exactly the same neighbors. This means that the underlying vertices whose names will be swapped will be identical under π and π'. Hence, swapping twice gives us back π. ∎

This fixes the problem we had because we can use π in the no case and π' in the yes case. This way in both cases the edge the algorithm sees will be (v, u). Note that the dual permutation is a function of the discovered graph. So if we are in another fixing then the dual permutation for π may not be π'.

We are now ready to define blocks such that $\mathcal{M}$ is block-compliant with respect to it. Let $\mathcal{X}_2$ be the subset of $\mathcal{X}_1$ where the set of star centers is S, the core indices in their adjacency lists are $\psi(S)$ with permutations $\sigma_C(S)$ and the labeling permutation is either π or π'. Define $\mathcal{Y}_2$ similarly. We will ensure that $\mathcal{M}$ is confined to $(\mathcal{X}_2, \mathcal{Y}_2)$. This makes sense because if the star centers, their adjacency lists or the labeling permutations are different, the observed graph may be different. However, edges of $\mathcal{M}$ will go from nodes that have permutation π in the no case to nodes that have permutation π' in the yes case and vice versa.

Claim 10.4.7. *For any node in $\mathcal{X}_2$ with labeling permutation π and any node in $\mathcal{Y}$ with labeling permutation π', the observed graph in the core G^C_{obs} is the same.*

Proof. Since $\mathcal{X}_2$ is a finer division of $\mathcal{X}_1$, any node in it has the core graph G^C_{NO}. Similarly, any node in $\mathcal{Y}_2$ has core graph G^C_{YES} which is the coupled graph of G^C_{NO}. The set of star centers S is the same for all nodes in $\mathcal{X}_2$ and $\mathcal{Y}_2$. Finally, the set of petals for a node in $\mathcal{X}_2$

Symbol / Term	Description
$P := P_{\text{NO}} \cup P_{\text{YES}}$	Set of petal vertices discovered in either the no or yes case under a fixed permutation π.
P_{pair}	Set of petal pairs (u, u') where $u \in P_{\text{NO}}$ and u' is its *corresponding petal* in P_{YES}.
π, π'	Labeling permutation π and its *dual permutation* π'; π' is obtained by swapping labels within petal pairs under labeling permutation π.
$\mathcal{X}_2, \mathcal{Y}_2$	Subsets of $\mathcal{X}_1, \mathcal{Y}_1$ with the following fixing: labeling permutations are π or π', star centers are S and the randomness of $\psi(S), \sigma_C(S)$ in the adjacency lists is fixed.
$\mathcal{X}_3, \mathcal{Y}_3$	Subsets of $\mathcal{X}_2, \mathcal{Y}_2$ with the following fixing: $\ell(C \setminus P)$ (which include edges to D), edges to D for queried locations for all $v \in P$, and edge to D for pairs $(u, u') \in P_{\text{pair}}$.
$\mathsf{SNL}_{u,u'}$	Shared neighbor list of petal pair (u, u')—unordered list of edges to D assigned to the pair (u, u') to be split later between u and u'.
$\mathsf{SNL}^{\text{obs}}_{u,u'}$	Discovered portion of the shared neighbor list via queries to D.

Table 10.6: Notation for Sections 10.4.2 and 10.4.3.

under π has the same labels as the set of petals for a node in $\mathcal{Y}_2$ under π' by definition of π'. Therefore, the observed graph in the core G^C_{obs} is the same for all these nodes. ∎

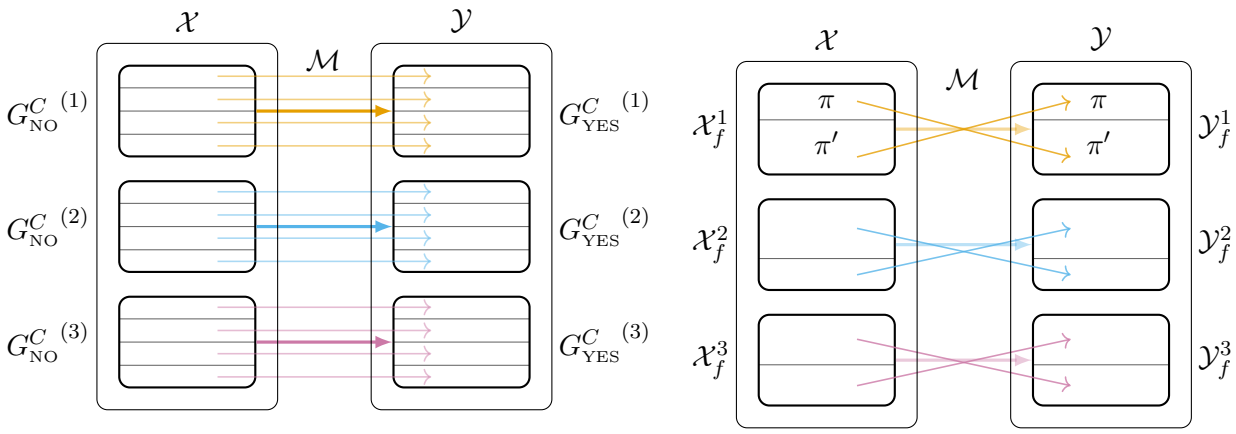
Fixings So Far

- Section 10.4.1: Fixed structure of core: G^C_{NO} in the no case and G^C_{YES} in the yes case.
- Section 10.4.2: Fixed set of star centers S along with $\psi(S)$ and $\sigma_C(S)$. Also, fixed the labeling permutations π and its dual π'.

10.4.3 Fixing edges to D and Adjacency Lists of Non-Petals

We are currently in a block where the structure in the core C is fixed, the set of star centers S is fixed along with $\psi(S), \sigma_C(S)$ and the labeling permutation is either π or π'.

We now describe the fixing we have for the edges to D. We fix the edges to D for all vertices in $C - P$ (the non-petal vertices in the core). For vertices in P we also fix edges to D for each petal pair in P_{pair} i.e. if $(u, u') \in P_{\text{pair}}$ is a pair then we fix edges to D for $u \cup u'$ but do not specify the exact endpoint. The number of neighbors to D we have to pick for $u \cup u'$ is $2d^* - n^\delta - 1$. This is because one petal vertex is in A and the other is



(a) This figure shows coupling of the core for graph $\mathcal{H}$ and how the matching stays block-compliant for the blocks and the final sub-blocks.

(b) This figure shows that the matching is block-compliant for the final blocks and how we pick the final edges of $\mathcal{M}$.

Figure 10.3: Showing the block-compliant matching $\mathcal{M}$.

in B, so they have n^δ and 1 core indices respectively. To do this we take the union of a set of $d^* - 1$ distinct neighbors in D with a set of $d^* - n^\delta$ distinct neighbors in D. If the intersection size of these two sets is outside the range $n^{1-\delta}/4 \pm 3\sqrt{\log n} \cdot n^{(1-\delta)/2}$ for any petal pair then the corresponding block is bad. Using Claim 10.4.8 and union bounding over all petal pairs we can say that this happens with probability $1/n$.

We call this unordered list the **shared neighbor list** of u and u' ($\mathsf{SNL}_{u,u'}$). We have that $|\mathsf{SNL}_{u,u'}| = 2d^* - n^\delta - 1$. When we later partition those edges between u and u', we will add these edges to their adjacency lists. Also, note that the maximum frequency of an element of D in $\mathsf{SNL}_{u,u'}$ is 2 because we have a union of two lists each of which contains distinct elements of D. If some element appears twice in $\mathsf{SNL}_{u,u'}$ then it will be a neighbor of both u and u'.

Claim 10.4.8. *Let S_1 be sample of D^R of size $d^* - 1$ without replacement and let S_2 be sample of D^R of size $d^* - n^\delta$ without replacement. The intersection size of S_1 and S_2 is concentrated in $n^{1-\delta}/4 \pm 3\sqrt{\log n} \cdot n^{(1-\delta)/2}$ with high probability.*

Proof. To show bounds on the intersection size we will fix the randomness of S_1 and then use the randomness of S_2 to get the bounds on the size. Let X_i be the random variable which is 1 if the i^{th} sample of S_2 lands in S_1 and 0 otherwise for $i \in [d^* - n^\delta]$.

Even though the draws are *without* replacement, every element of the ground set is equally likely to appear in position i. Hence, we have $\mathbb{E}[X_i] = \Pr(X_i = 1) = \frac{d^*-1}{n^{1-\delta}}$. Let $X = \sum_i X_i$ be the random variable denoting the intersection size. Using linearity of expectation we get:

$$\mathbb{E}[X] = \sum_{i=1}^{d^*-n^\delta} \mathbb{E}[X_i] = (d^* - n^\delta) \cdot \frac{d^*-1}{n^{1-\delta}} = n^{1-\delta}/4 - n^\delta/2 - 1/2 + n^{2\delta-1}.$$

Since elements in S_2 are sampled uniformly without replacement, the variables $\{X_i\}$ are negatively associated. Since each X_i is in $[0,1]$, we can apply the Chernoff bound in Proposition 2.4.3 with $\varepsilon = 10\sqrt{\log n / n^{1-\delta}}$:

$$\begin{aligned}\Pr(|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]) &\leqslant 2\exp\left(-\varepsilon^2 \cdot \mathbb{E}[X]/4\right)\\ &\leqslant 2\exp\left(-\varepsilon^2 \cdot n^{1-\delta}/20\right)\\ &\leqslant 2\exp\left(-100\log n/20\right)\\ &\leqslant n^{-4}.\end{aligned}$$

Therefore, with high probability, X is in $n^{1-\delta}/4 \pm 3\sqrt{\log n} \cdot n^{(1-\delta)/2}$. ∎

We also fix the adjacency list $\ell(v)$ for all $v \in V - P$. $\psi(v), \sigma_C(v)$ have already been fixed for $v \in S$ so we only fix $\sigma_D(v)$ for $v \in S$. We now also fix this for other vertices in $C - P - S$. The only vertices in $C - P - S$ are the isolated vertices in E^C_{obs} (all queries on them go to D). We fix their entire adjacency lists. Note that we can fix $\ell(D)$ because for all $v \in D$, their edges have been fixed (up to switching within petal pairs). So their adjacency lists are not completely fixed because they may have an edge to petal pair (u, u') which implies that the endpoint of the edge is not fixed. However, if it has two edges to a petal pair (u, u') then we arbitrarily fix one of neighbors as u and the other one as u'. Note that any edge to a petal pair (u, u') corresponds to a fixed element in $\mathsf{SNL}_{u,u'}$. So it is a function of whether that particular neighbor is assigned to u or u'. If this fixing violates Claim 10.3.2 then the corresponding block is bad, and we ignore it. We can do this because the degrees of all vertices in D have been fixed.

Fixing $\ell(D)$ fixes the discovered edges from D to P but the endpoints of these edges are not fixed within the petal pairs. These discovered edges are present in the shared neighbor list SNL. For $(u, u') \in P_{\text{pair}}$, $\mathsf{SNL}^{\text{obs}}_{u,u'}$ represents the *observed* part of the shared neighbor list of (u, u'). The only queried edges that are not fixed are the ones in P. Thus, for vertices in P, we now fix edges to D for all the queried locations in their adjacency lists. We will do this by picking neighbors from SNL. These can be the observed elements ($\mathsf{SNL}^{\text{obs}}_{u,u'}$), the duplicate elements, or remaining ones which are not observed and have just one copy.

Consider vertex $u \in P$ in the no case where labeling permutation π is used. The vertex u has q_u queried locations out of d^* locations. The vertex $u \in P$ in the yes case where labeling permutation π' is used, is a different vertex in terms of its name in the underlying structure, but it has the same set of q_u out of d^* queried indices. Thus, we fix the edges to D in these locations for all petals in P. After this the size of $\mathsf{SNL}_{u,u'}$ is $\mathsf{snl}_{u,u'} := 2d^* - 1 - n^\delta - q_u - q_{u'}$. Note that when fixing these edges to D we also have to fix their *order*. This fixes some randomness of $\sigma_D(P)$.

We are now ready to define blocks such that $\mathcal{M}$ is block-compliant with respect to it. Let $\mathcal{X}_3$ be the subset of $\mathcal{X}_2$ with all the above fixings and define $\mathcal{Y}_3$ similarly. We will ensure that $\mathcal{M}$ is confined to $(\mathcal{X}_3, \mathcal{Y}_3)$. All the edges of the graph have been fixed except for endpoints of some edges that can switch within petal pairs. Note that since the queries are fixed, this also fixes the discovered edges from D. The discovered edges from D to $C - P$ are fixed and the discovered edges to P are fixed in the shared neighbor lists. The edges in $\mathsf{SNL}^{\text{obs}}_{u,u'}$ will be part of the observed graph irrespective of whether they are assigned to u or u'. We now show that the observed graph is almost fixed within the blocks.

Claim 10.4.9. *For any node in $\mathcal{X}_3$ with permutation π and any node in $\mathcal{Y}_3$ with permutation π', the observed graph G_{obs} is the same up to edges from D to P switching within petal pairs. Also, the discovered edges are fixed in the shared neighbor lists.*

Proof. We know from Claim 10.4.7 that the observed graph in the core G^C_{obs} is the same for these nodes. For all vertices in $C - P$, we have fixed their edges to D and their adjacency lists. Since the queries are deterministically fixed, all the discovered edges from these vertices will be identical for all nodes under consideration. For vertices in P, we have fixed the edges in the queried locations implying that all the discovered edges from the petal vertices will be identical for all nodes under consideration.

Finally, for vertices in D, their adjacency lists have been fixed except some edges are to petal pairs instead of the explicit vertex within the petal pair. However, the discovered edges are fixed in the shared neighbor lists for petal pairs (specific positions of $\mathsf{SNL}_{u,u'}$ are part of the discovered edges irrespective of whether they are assigned to u or u'). Therefore, G_{obs} is the same for all nodes under consideration up to edges from D to P switching within petal pairs. ∎

Fixings So Far

- Section 10.4.1: Fixed structure of core: G^C_{NO} in the no case and G^C_{YES} in the yes case.
- Section 10.4.2: Fixed set of star centers S along with $\psi(S), \sigma_C(S)$ and labeling permutations π, π'.
- Section 10.4.3: Fixed edges to D for all vertices in $C - P$ and for vertices in P fixed edges to D for petal pairs using SNL. Fixed $\ell(V - P)$ except for neighbors of D in P which can switch within petal pairs. Finally, fixed all queried indices in $\ell(P)$.

10.4.4 Fixing Final Blocks

We now fix what the final blocks are. We know that the discovered edges from D to P are fixed but the endpoints of these edges are not fixed within the petal pairs (Claim 10.4.9). However, these discovered edges are fixed in the shared neighbor list. So for each petal pair we fix the endpoints for these discovered edges ($\mathsf{SNL}^{\text{obs}}_{u,u'}$) by partitioning the discovered edges in the shared neighbor lists between the petals in the petal pairs. Note that this could be any arbitrary partitioning for the observed edges $\mathsf{SNL}^{\text{obs}}_{u,u'}$ for all $(u, u') \in P_{\text{pair}}$. We let $\mathsf{SNL}^{\text{obs}}_{u,u'}(u)$ be the subset of $\mathsf{SNL}^{\text{obs}}_{u,u'}$ assigned to u with size $\mathsf{snl}^{\text{obs}}_u$ and let $\mathsf{SNL}^{\text{obs}}_{u,u'}(u')$ be the subset of $\mathsf{SNL}^{\text{obs}}_{u,u'}$ assigned to u' with size $\mathsf{snl}^{\text{obs}}_{u'}$.

If this fixing violates Claim 10.3.12 then the corresponding block is bad, and we ignore it. We can do this because the discovered edges from D have been fixed. We are now ready to define the final blocks such that $\mathcal{M}$ is block-compliant with respect to it. Let $\mathcal{X}_f$ be the subset of $\mathcal{X}_3$ with this fixing that partitioning of discovered edges in the shared neighbor lists of the petals, and define $\mathcal{Y}_f$ similarly. We will ensure that $\mathcal{M}$ is confined to $(\mathcal{X}_f, \mathcal{Y}_f)$. This final fixing has now completely fixed the observed graph:

Claim 10.4.10. *For any node in $\mathcal{X}_f$ with permutation π and any node in $\mathcal{Y}_f$ with permutation π', the observed graph G_{obs} is the same.*

Proof. We know from Claim 10.4.9 that the observed graph G_{obs} is the same up to edges from D to P switching within petal pairs. Fixing the partitioning of the queried edges from D in $\mathsf{SNL}_{u,u'}$ for all $(u, u') \in P_{\text{pair}}$ fixed all the discovered edges from D to P (this has removed all the randomness of endpoints of edges switching within petal pairs). Thus, all the edges that are discovered by Alg are now same among all nodes in $\mathcal{X}_f$ under π and $\mathcal{Y}_f$ under π' making G_{obs} the same. ∎

Symbol / Term	Description
$\mathcal{X}_f, \mathcal{Y}_f$	Final blocks of nodes in $\mathcal{X}$ and $\mathcal{Y}$, obtained by taking subsets of $\mathcal{X}_3$ and $\mathcal{Y}_3$ with the following fixing: For all $(u,u') \in P_{\text{pair}}$, arbitrarily split up $\mathsf{SNL}^{\text{obs}}_{u,u'}$ between u and u'.
$\mathcal{X}_f^\pi, \mathcal{X}_f^{\pi'}, \mathcal{Y}_f^\pi, \mathcal{Y}_f^{\pi'}$	Subsets of $\mathcal{X}_f$ and $\mathcal{Y}_f$ where labeling permutations are fixed to either π or π'.
$\mathsf{SNL}^{\text{obs}}_{u,u'}(u)$	Subset of observed edges from the shared neighbor list $\mathsf{SNL}_{u,u'}$ assigned to petal u. $\mathsf{snl}^{\text{obs}}_u := \left\lvert \mathsf{SNL}^{\text{obs}}_{u,u'}(u) \right\rvert$

Table 10.7: Notation for Section 10.4.4.

Fixings So Far

- Section 10.4.1: Fixed structure of core: G^C_{NO} in the no case and G^C_{YES} in the yes case.
- Section 10.4.2: Fixed set of star centers S along with $\psi(S), \sigma_C(S)$ and labeling permutations π, π'.
- Section 10.4.3: Fixed edges to D for $C - P$ and for P fixed edges to D for petal pairs using SNL. Fixed $\ell(V - P)$ except for neighbors of D in P which can switch within petal pairs. Finally, fixed all queried indices in $\ell(P)$.
- Section 10.4.4: Fixed the split of observed edges in SNL ($\mathsf{SNL}^{\text{obs}}_{u,u'}$) between vertices in petal pairs. This fixes the entire observed graph.

Randomness left: Splitting remaining edges in SNL and placing them in the adjacency lists of the petals. Note that the petals only have elements in the queried indices and all other indices are empty.

The observed graph G_{obs} is the same for any node in $\mathcal{X}_f$ with permutation π and any node in $\mathcal{Y}_f$ with permutation π'. The same proof also shows that the observed graph G_{obs} is the same for any node in $\mathcal{X}_f$ with permutation π' and any node in $\mathcal{Y}_f$ with permutation π. We let $\mathcal{X}_f^\pi$ represent the part of block with permutation π and $\mathcal{X}_f^{\pi'}$ represent the part of block with permutation π'. Define $\mathcal{Y}_f^\pi$ and $\mathcal{Y}_f^{\pi'}$ similarly. The only randomness left now is in partitioning the shared neighbor list between the petals and $\psi(v), \sigma_C(v)$ and $\sigma_D(v)$ for all $v \in P$. We now show that the sizes of the blocks $\mathcal{X}_f$ and $\mathcal{Y}_f$ are the same.

Claim 10.4.11. $\left|\mathcal{X}_f^\pi\right| = \left|\mathcal{Y}_f^\pi\right|$, $\left|\mathcal{X}_f^{\pi'}\right| = \left|\mathcal{Y}_f^{\pi'}\right|$ *and* $|\mathcal{X}_f| = |\mathcal{Y}_f|$.

Proof. Consider labeling permutation π and any petal pair $(u,u') \in P_{\text{pair}}$ where $u' \in A$

and $u \in B$. The following calculations will be the same in the yes and the no case under the same permutation because the partitioning and picking core indices does not depend on it.

Under labeling permutation π, u has one core index and u' has n^δ core indices. Also, the q_u queried positions of u and the $q_{u'}$ positions of u' are already filled. Thus, u needs $d^* - 1 - q_u$ elements of $\mathsf{SNL}_{u,u'}$ and u' needs the remaining $d^* - n^\delta - q_{u'}$. We have already assigned $\mathsf{snl}_u^{\text{obs}}$ neighbors to u and $\mathsf{snl}_{u'}^{\text{obs}}$ neighbors to u' from the shared neighbor list (apart from the q_u and $q_{u'}$ elements that have already been put in the adjacency lists of u and u'). Thus, we need to assign $d^* - 1 - q_u - \mathsf{snl}_u^{\text{obs}}$ neighbors to u out of $\mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}}$ neighbors where $\mathsf{snl}_{u,u'}^{\text{obs}} := \left|\mathsf{SNL}_{u,u'}^{\text{obs}}\right|$. We need to be careful here because duplicate elements ($\mathsf{SNL}_{u,u'}^{\text{dup}}$) have to be split between u and u'. Some of them were already placed in the queried positions of u and u' so the remaining ones have count $\mathsf{snl}_{u,u'}^{\text{dup}} = \mathsf{snl}_u^{\text{dup}} + \mathsf{snl}_{u'}^{\text{dup}}$. Thus, under permutation π, the number of choices to pick the remaining dummy neighbors of u is $\binom{\mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}} - \mathsf{snl}_{u,u'}^{\text{dup}}}{d^* - 1 - q_u - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_u^{\text{dup}}}$.

We also have to choose the core indices for u and u' which can be done in $d^* - q_u$ ways for u and $\binom{d^* - q_{u'}}{n^\delta}$ ways for u'. Fixing this fixes the randomness of $\psi(u), \psi(u')$. The number of permutations $\sigma_C(u')$ is $(n^\delta)!$ and $\sigma_C(u)$ is 1. The number of permutations $\sigma_D(u')$ is $(d^* - n^\delta - q_{u'})!$ and $\sigma_D(u)$ is $(d^* - 1 - q_u)!$. This fixes the randomness of σ_C, σ_D along with ψ.

We note that all of these steps are independent of each other. This implies that the number of fixings for pair (u, u') is

$$f_{u,u'}^{\pi} := \binom{\mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}} - \mathsf{snl}_{u,u'}^{\text{dup}}}{d^* - 1 - q_u - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_u^{\text{dup}}} \left[\binom{d^* - q_{u'}}{n^\delta}(d^* - q_u)(n^\delta)!(d^* - 1 - q_u)!(d^* - n^\delta - q_{u'})!\right].$$

Thus, the size of $\mathcal{X}_f^\pi$ and $\mathcal{Y}_f^\pi$ is:

$$\prod_{(u,u') \in P_{\text{pair}}} \binom{\mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}} - \mathsf{snl}_{u,u'}^{\text{dup}}}{d^* - 1 - q_u - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_u^{\text{dup}}} \left[\binom{d^* - q_{u'}}{n^\delta}(d^* - q_u)(n^\delta)!(d^* - 1 - q_u)!(d^* - n^\delta - q_{u'})!\right].$$

We can do a similar process under the permutation π'. The only difference there is that $u \in A$ and $u' \in B$. Thus, we get that the size of $\mathcal{X}_f^{\pi'}$ and $\mathcal{Y}_f^{\pi'}$ is:

$$\prod_{(u,u') \in P_{\text{pair}}} \binom{\mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}} - \mathsf{snl}_{u,u'}^{\text{dup}}}{d^* - 1 - q_{u'} - \mathsf{snl}_{u'}^{\text{obs}} - \mathsf{snl}_{u'}^{\text{dup}}} \left[\binom{d^* - q_u}{n^\delta}(d^* - q_{u'})(n^\delta)!(d^* - 1 - q_{u'})!(d^* - n^\delta - q_u)!\right].$$

These also imply that $|\mathcal{X}_f| = |\mathcal{Y}_f|$. ∎

Recall that $\mathcal{X}_f$ with π has the same observed graph as $\mathcal{Y}_f$ with π' (Claim 10.4.10). The same proof also gives us that $\mathcal{X}_f$ with π' has the same observed graph as $\mathcal{Y}_f$ with π. Thus, we need to add edges of $\mathcal{M}$ between $\mathcal{X}_f^{\pi}$ and $\mathcal{Y}_f^{\pi'}$ (similarly for $\mathcal{X}_f^{\pi'}$ and $\mathcal{Y}_f^{\pi}$). We will add an arbitrary matching between $\mathcal{X}_f^{\pi}$ and $\mathcal{Y}_f^{\pi'}$ which is perfect on the smaller side (see Figure 10.3). This means that the size of the matching will be $\min\left(\left|\mathcal{X}_f^{\pi}\right|, \left|\mathcal{Y}_f^{\pi'}\right|\right)$. We will get a matching of the same size between $\mathcal{X}_f^{\pi'}$ and $\mathcal{Y}_f^{\pi}$. We now have to show that the size of the matching is large enough. To show that we will show that these sets are of a comparable size. To prove this claim we first show that for any pair $(u, u') \in P_{\text{pair}}$ their number of possible fixings are similar under π and π'.

Claim 10.4.12. *We have* $\left|\mathcal{Y}_f^{\pi'}\right| = (1 \pm o(1))\left|\mathcal{X}_f^{\pi}\right|$.

Proof. Without loss of generality let $\left|\mathcal{X}_f^{\pi}\right| \geqslant \left|\mathcal{Y}_f^{\pi'}\right|$. Then we have to show $\left|\mathcal{Y}_f^{\pi'}\right| \geqslant (1 - o(1))\left|\mathcal{X}_f^{\pi}\right|$. We will first bound $f_{u,u'}^{\pi'}/f_{u,u'}^{\pi}$ for any $(u, u') \in P_{\text{pair}}$ and then use that to bound $\left|\mathcal{Y}_f^{\pi'}\right| / \left|\mathcal{X}_f^{\pi}\right|$. We have the following bounds:

- $|\mathsf{SNL}_{u,u'}| = \mathsf{snl}_{u,u'} = 2d^* - q_u - q_{u'} - n^{\delta} - 1$.
- $\mathsf{snl}_u^{\text{obs}}, \mathsf{snl}_{u'}^{\text{obs}} \leqslant 4n^{\varepsilon}$ (by Claim 10.3.12).
- $\left|\mathsf{SNL}_{u,u'}^{\text{obs}}\right| = \mathsf{snl}_{u,u'}^{\text{obs}} = \mathsf{snl}_u^{\text{obs}} + \mathsf{snl}_{u'}^{\text{obs}}$ since the observed edges belong to either u or u'.
- $n^{1-\delta}/2 - 6\sqrt{\log n} \cdot n^{(1-\delta)/2} - q_u - q_{u'} \leqslant \mathsf{snl}_{u,u'}^{\text{dup}} \leqslant n^{1-\delta}/2 + 6\sqrt{\log n} \cdot n^{(1-\delta)/2}$.
- $n^{1-\delta}/4 - 3\sqrt{\log n} \cdot n^{(1-\delta)/2} - q_u \leqslant \mathsf{snl}_u^{\text{dup}} \leqslant n^{1-\delta}/4 + 3\sqrt{\log n} \cdot n^{(1-\delta)/2}$ (similar for $\mathsf{snl}_{u'}^{\text{dup}}$).
- $\alpha = \log(n^{\varepsilon} \log n)$ and $\beta = \log(16n^{2\varepsilon+\delta} \log^2 n)$ (defined in Lemma 10.3.11).

We define the following:

$$\begin{aligned}
\kappa &:= \mathsf{snl}_{u,u'} - \mathsf{snl}_{u,u'}^{\text{obs}} - \mathsf{snl}_{u,u'}^{\text{dup}} \leqslant 2d^*, \\
\lambda_u &:= d^* - 1 - q_u - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_u^{\text{dup}}, \\
\lambda_{u'} &:= d^* - 1 - q_{u'} - \mathsf{snl}_{u'}^{\text{obs}} - \mathsf{snl}_{u'}^{\text{dup}}, \\
\zeta &:= 3\sqrt{\log n} \cdot n^{(1-\delta)/2}.
\end{aligned}$$

The values of $f^{\pi}_{u,u'}$ and $f^{\pi'}_{u,u'}$ are:

$$f^{\pi}_{u,u'} := \binom{\mathsf{snl}_{u,u'} - \mathsf{snl}^{\text{obs}}_{u,u'} - \mathsf{snl}^{\text{dup}}_{u,u'}}{d^* - 1 - q_u - \mathsf{snl}^{\text{obs}}_{u} - \mathsf{snl}^{\text{dup}}_{u}} \left[\binom{d^* - q_{u'}}{n^\delta} (d^* - q_u)(n^\delta)!(d^* - 1 - q_u)!(d^* - n^\delta - q_{u'})! \right].$$

$$f^{\pi'}_{u,u'} := \binom{\mathsf{snl}_{u,u'} - \mathsf{snl}^{\text{obs}}_{u,u'} - \mathsf{snl}^{\text{dup}}_{u,u'}}{d^* - 1 - q_{u'} - \mathsf{snl}^{\text{obs}}_{u'} - \mathsf{snl}^{\text{dup}}_{u'}} \left[\binom{d^* - q_{u}}{n^\delta} (d^* - q_{u'})(n^\delta)!(d^* - 1 - q_{u'})!(d^* - n^\delta - q_{u})! \right].$$

Our goal is to lower bound the ratio $r := \frac{f^{\pi'}_{u,u'}}{f^{\pi}_{u,u'}}$. To do this, we decompose r into three parts: $r = r_1 \cdot r_2 \cdot r_3$, where:

- r_1 is the ratio of the first binomial terms:

$$r_1 := \frac{\binom{\kappa}{\lambda_{u'}}}{\binom{\kappa}{\lambda_u}},$$

- r_2 is the ratio of the second binomial terms and the last two terms:

$$r_2 := \frac{\binom{d^* - q_u}{n^\delta}}{\binom{d^* - q_{u'}}{n^\delta}} \cdot \frac{(d^* - 1 - q_{u'})!(d^* - n^\delta - q_u)!}{(d^* - 1 - q_u)!(d^* - n^\delta - q_{u'})!}.$$

- r_3 is the ratio of the remaining two terms:

$$r_3 := \frac{(d^* - q_{u'}) \cdot (n^\delta)!}{(d^* - q_u) \cdot (n^\delta)!}.$$

We will lower bound each of them separately to get a bound on r. We first show that κ is always between $d^* - o(d^*)$ and $d^* + 2\zeta$. First consider the lower bound:

$$\kappa = 2d^* - q_u - q_{u'} - n^\delta - 1 - \mathsf{snl}_{u,u'} - \mathsf{snl}^{\text{dup}}_{u,u'} \geqslant 2d^* - 2^{\beta+1} - 1 - n^\delta - 8n^\varepsilon - d^* - 2\zeta \geqslant d^* - 3 \cdot 2^\beta - 2\zeta \geqslant d^* - o(d^*)$$

where the last inequality holds because $2^\beta + \zeta = o(d^*)$ (Eq (10.2)). We now show the upper bound:

$$\kappa = 2d^* - q_u - q_{u'} - n^\delta - 1 - \mathsf{snl}_{u,u'} - \mathsf{snl}^{\text{dup}}_{u,u'} \leqslant 2d^* - q_u - q_{u'} - (d^* - 2\zeta - q_u - q_{u'}) \leqslant d^* + 2\zeta$$

where the last inequality holds because $\zeta = o(d^*)$. We now start by lower bounding r_1:

$$r_1 = \binom{\kappa}{\lambda_{u'}} \binom{\kappa}{\lambda_u}^{-1}$$

$$
\begin{aligned}
&= \frac{\lambda_u! \cdot (\kappa - \lambda_u)!}{\lambda_{u'}! \cdot (\kappa - \lambda_{u'})!} \\
&= \frac{\lambda_u! \cdot (\lambda_{u'} - (n^\delta - 1))!}{(\lambda_u - (n^\delta - 1))! \cdot (\lambda_{u'})!} && \text{(By definition of } \lambda_u, \lambda_{u'}) \\
&= \frac{(\lambda_u) \cdot (\lambda_u - 1) \dots (\lambda_u - n^\delta + 2)}{(\lambda_{u'}) \cdot (\lambda_{u'} - 1) \dots (\lambda_{u'} - n^\delta + 2)} \\
&\geqslant \left(\frac{\lambda_u - n^\delta + 2}{\lambda_{u'}} \right)^{n^\delta - 1} \\
&\geqslant \left(\frac{d^* - q_u - n^\delta - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_u^{\text{dup}}}{d^* - \mathsf{snl}_{u'}^{\text{dup}}} \right)^{n^\delta} \\
&\geqslant \left(\frac{d^* - q_u - n^\delta - \mathsf{snl}_u^{\text{obs}} - \mathsf{snl}_{u'}^{\text{dup}} - q_{u'}}{d^* - \mathsf{snl}_{u'}^{\text{dup}}} \right)^{n^\delta} && (\mathsf{snl}_u^{\text{dup}} \leqslant \mathsf{snl}_{u'}^{\text{dup}} + q_{u'}) \\
&= \left(1 - \frac{q_u + q_{u'} + n^\delta + 4n^\varepsilon}{d^* - \mathsf{snl}_{u'}^{\text{dup}}} \right)^{n^\delta} && (\mathsf{snl}_u^{\text{obs}} \leqslant 4n^\varepsilon) \\
&\geqslant \exp\left(-16(q_u + q_{u'} + n^\delta + 4n^\varepsilon) \cdot n^{2\delta - 1} \right) && \text{(Since } 1 - x \geqslant \exp(-2x) \text{ for } x \in [0, 1/2])
\end{aligned}
$$

where the last inequality holds because $d^* - \mathsf{snl}_{u'}^{\text{dup}} \geqslant d^*/4$. We now consider r_2:

$$
\begin{aligned}
r_2 &= \frac{\binom{d^* - q_u}{n^\delta}}{\binom{d^* - q_{u'}}{n^\delta}} \cdot \frac{(d^* - 1 - q_{u'})! \cdot (d^* - n^\delta - q_u)!}{(d^* - 1 - q_u)! \cdot (d^* - n^\delta - q_{u'})!} \\
&= \frac{(d^* - q_u)! \cdot (d^* - q_{u'} - n^\delta)!}{(d^* - q_{u'})! \cdot (d^* - q_u - n^\delta)!} \cdot \frac{(d^* - 1 - q_{u'})! \cdot (d^* - n^\delta - q_u)!}{(d^* - 1 - q_u)! \cdot (d^* - n^\delta - q_{u'})!} \\
&= \frac{d^* - q_u}{d^* - q_{u'}}.
\end{aligned}
$$

Notice that r_3 is exactly the reciprocal of r_2:

$$
r_3 = \frac{(d^* - q_{u'}) \cdot (n^\delta)!}{(d^* - q_u) \cdot (n^\delta)!} = \frac{d^* - q_{u'}}{d^* - q_u}.
$$

Thus, we get $r_2 \cdot r_3 = 1$. This happens because in both cases we have one adjacency list with q_u filled slots and the other with $q_{u'}$ filled slots. Therefore, the number of permutations stays the same. Hence, we get $r = r_1 = \exp(-8(q_u + q_{u'} + n^\delta + 4n^\varepsilon) \cdot n^{2\delta - 1})$. Note that q_u

and $q_{u'}$ are the only terms that depend on the petal pair. The final part is to show that $r_f := \left|\mathcal{Y}_f^{\pi'}\right| \left(\left|\mathcal{X}_f^{\pi}\right|\right)^{-1} \geqslant (1 - o(1))$:

$$\begin{aligned}
r_f &= \left|\mathcal{Y}_f^{\pi'}\right| \left(\left|\mathcal{X}_f^{\pi}\right|\right)^{-1} \\
&= \prod_{(u,u') \in P_{\text{pair}}} f_{u,u'}^{\pi'} / f_{u,u'}^{\pi} \\
&\geqslant \exp\left(- \sum_{(u,u') \in P_{\text{pair}}} 16(q_u + q_{u'} + n^{\delta} + 4n^{\varepsilon}) \cdot n^{2\delta - 1} \right) \\
&\geqslant \exp\left(-16n^{3\delta-1} - 64n^{2\delta+\varepsilon-1} - 16n^{2\delta-1} \sum_{(u,u') \in P_{\text{pair}}} q_u + q_{u'} \right)
\end{aligned}$$

To prove this, we need to upper bound the sum of queries over petals. We partition P_{pair} into different chunks based on the number of queries on the petals. For all $i \in [\alpha, \beta]$ chunk i contains all the petal pairs (u, u') with q_u between 2^i and 2^{i+1}. We also define chunk 0 to contain all the petal pairs (u, u') with q_u at most 2^{α}. The number of petal pairs in chunk i is at most $2^{-i} \cdot 8n^{2\varepsilon+\delta} \log^2 n$ since that is an upper bound on the number of 2^i-heavy petal vertices Lemma 10.3.11. For chunk i we have the following:

$$s^{(i)} := \sum_{(u,u') \in \text{chunk } i} q_u \leqslant 2^{i+1} \cdot (2^{-i} \cdot 2^{\beta-1}) = 2^{\beta}.$$

For chunk 0 we bound the number of queries with 2^{α} and the number of petal pairs is at most $4n^{\varepsilon+\delta} \log n$. This is because that is the bound on the number of discovered edges (Claim 10.3.6). This gives us a bound for chunk 0:

$$s^{(0)} := \sum_{(u,u') \in \text{chunk } 0} q_u \leqslant 2^{\alpha} \cdot 4n^{\varepsilon+\delta} \log n.$$

This gives us $\sum_{(u,u') \in P_{\text{pair}}} q_u \leqslant 2^{\alpha+2} \cdot n^{\varepsilon+\delta} \log n + 2^{\beta} \log n$. Thus, we finally have:

$$\begin{aligned}
r_f &\geqslant \exp\left(-16n^{3\delta-1} - 64n^{2\delta+\varepsilon-1} - 16n^{2\delta-1} \sum_{(u,u') \in P_{\text{pair}}} q_u + q_{u'} \right) \\
&\geqslant \exp\left(-16n^{3\delta-1} - 64n^{2\delta+\varepsilon-1} - 16n^{2\delta-1} \cdot (2^{\beta+1} \log n + 2^{\alpha+3} \cdot n^{\varepsilon+\delta} \log n) \right) \\
&= \exp\left(-\Theta(n^{3\delta-1} + n^{2\delta+\varepsilon-1} + n^{3\delta+2\varepsilon-1} \log^3 n) \right)
\end{aligned}$$

$$= (1 - o(1))$$

where the last inequality holds because of Eq (10.2). ∎

This finally implies that the matching $\mathcal{M}$ has a large size.

Proof of Lemma 10.4.3. We divide $\mathcal{X}$ and $\mathcal{Y}$ into blocks by fixing some part of the randomness. $\mathcal{X}_f$ and $\mathcal{Y}_f$ represent blocks under some fixing. We add an arbitrary matching between $\mathcal{X}_f^{\pi}$ and $\mathcal{Y}_f^{\pi'}$ of size $\min(\left|\mathcal{X}_f^{\pi}\right|, \left|\mathcal{Y}_f^{\pi'}\right|)$. Similarly, we add an arbitrary matching between $\mathcal{X}_f^{\pi'}$ and $\mathcal{Y}_f^{\pi}$ of size $\min(\left|\mathcal{X}_f^{\pi'}\right|, \left|\mathcal{Y}_f^{\pi}\right|)$. These describe the edges of $\mathcal{M}$.

The size of $\mathcal{M}$ is exactly equal to the number of matched nodes in $\mathcal{X}$. We have ignored $(1 - o(1))$ fraction of the nodes in $\mathcal{X}$ because of high probability events. For the remaining nodes, each of them belong in some final block $\mathcal{X}_f$. For every such block we have matched between $|\mathcal{X}_f|$ and $(1 - o(1)) |\mathcal{X}_f|$ nodes. Thus, $|\mathcal{M}| \geqslant (1 - o(1)) \cdot (1 - o(1)) |\mathcal{X}| = (1 - o(1)) |\mathcal{X}|$. ∎

10.5 The Algorithm

In this section, we give an algorithm that gives an n^{δ}-approximation to the maximum matching size using $O(n^{2-2\delta} \log n)$ non-adaptive queries. In particular, this gives us a $\sqrt{n}$-approximation algorithm in $O(n \log n)$ queries.

Theorem 31. *There is a randomized algorithm that with probability $1 - 1/n$ can output an n^{δ}-approximation to the matching size using $O(n^{2-2\delta} \log n)$ non-adaptive queries in the adjacency list model for any constant $0 < \delta < 1$. In particular, when $\delta = 1/2$, the algorithm gives a $\sqrt{n}$-approximation to the matching size in $O(n \log n)$ queries.*

Consider the following algorithm:

Algorithm 21. An n^{δ}-approximation Algorithm.

Input: Non-Adaptive Query access to the Adjacency List of a graph $G = (V, E)$.

Output: An n^{δ}-approximation to the maximum matching size of G.

1. Query $q := 80n^{1-2\delta} \log n$ random neighbors for each vertex in V.
2. If $q < 1$ then sample a random neighbor with probability q.

3. Output the size of the largest matching M in the returned edges.

Before going into the analysis we note that since we are in the non-adaptive model, and we do not know vertex degrees, our algorithm is allowed to query random neighbors of a vertex in the adjacency list. It is very easy to get rid of this assumption if we increase the number of queries by a factor of $O(\log n)$. We guess the size of the adjacency list of a vertex in powers of 2 and then make the random queries assuming that the degree of the vertex is exactly the guess. Let the true size of the list be between 2^{i-1} and 2^i. When the guess is 2^i, a random query lands in the NULL part with probability at most $1/2$. So a constant fraction of the queries will be random queries over the true adjacency list. Thus, we have simulated random query access with an $O(\log n)$ factor overhead.

We now turn to the analysis. Consider a graph $G = (V, E)$ and let M^* be a maximum matching of G. The main lemma we will prove in this section is the following:

Lemma 10.5.1. $|M| \geqslant n^{-\delta} |M^*|$.

If $|M^*| \leqslant n^\delta$ then we get an n^δ-approximation easily by outputting a single edge of the graph. We will be able to find one edge because we query all the vertices. Thus, for the rest of the proof, we assume $|M^*| \geqslant n^\delta$. We define V_ℓ to be the low degree vertices in $V(M^*)$

$$V_\ell := \{v \in V(M^*) \mid \deg(v) \leqslant 4n^{1-\delta}\}$$

and define $V_h := V(M^*) - V_\ell$ to be the high degree vertices in $V(M^*)$. We break the rest of the analysis into two cases.

- **Case 1:** At least $|M^*|/2$ of the vertices of M^* have degree at most $4n^{1-\delta}$ i.e. $|V_\ell| \geqslant |M^*|/2$.
  In this case, we will find a large fraction of the edges of M^*.
- **Case 2:** At least $|M|/2$ of the vertices of M^* have degree strictly larger than $4n^{1-\delta}$ i.e. $|V_h| \geqslant |M^*|/2$. In this case, we will find many edges because the degrees of the vertices are large.

We first consider Case 1 and show that the matching found is large enough.

Claim 10.5.2. *If $|V_\ell| \geqslant |M^*|/2$, then $|M| \geqslant |M^*| \cdot n^{-\delta}$ with probability at least $1 - 1/n$.*

Proof. Consider a vertex $v \in V_\ell$ and let its edge in the maximum matching be called its *special edge*. We make q queries on the vertex v. The queries find the special edge of v with probability $q/\deg(v) \geqslant 20n^{-\delta}\log n$. If $q < 1$ then we make a query with probability q and find the special edge with probability $1/\deg(v)$ getting the same bound. This holds true for all vertices in V_ℓ.

We know that $|V_\ell| \geqslant |M^*|/2$ so we fix any $|M^*|/2$ vertices in the set and define the following. Let X_i be the random variable which is 1 if the queries find the special edge for the i^{th} vertex in V_ℓ and 0 otherwise, for $i \in [|M^*|/2]$. We know that $\mathbb{E}[X_i] = \Pr(X_i = 1) \geqslant 20n^{-\delta}\log n$. Let $X = \sum_{i=1}^{|M^*|/2} X_i$ be the number of special edges found. We have that $X \leqslant |M|$ because we are only looking at a subset of the edges found. The expected value of X is $\mathbb{E}[X] = \sum_i \mathbb{E}[X_i] \geqslant 20n^{-\delta}\log n \cdot |M^*|/2 = \mu_L$. Each X_i is independent and in $[0,1]$ so we can apply the Chernoff bound (Proposition 2.4.3) with $\varepsilon = 1/2$:

$$\Pr(X < \mu_L/2) \leqslant \exp(-\mu_L/10) \leqslant \exp(-n^{-\delta}\log n \cdot |M^*|) \leqslant n^{-1}.$$

Thus, $|M| \geqslant X \geqslant \mu_L/2 = 5n^{-\delta}\log n \cdot |M^*|$ with probability $1 - 1/n$. ∎

We now consider Case 2 and show that the matching found is large enough in this case too.

Claim 10.5.3. *If $|V_\ell| < |M|/2$, then $|M| \geqslant |M^*| \cdot n^{-\delta}$ with probability at least $1 - 1/\mathrm{poly}(n)$.*

Proof. In this case, we have at least $|M^*|/2$ vertices in V_h. We now consider the following procedure to find a large matching. We go over the vertices of V_h in an arbitrary but fixed order and maintain a matching in the following inductive way. We start with an empty matching M_0. For the i^{th} vertex in the order, we construct matching M_i by greedily trying to increase the size of M_{i-1} by adding an edge to it from the queried edges for the i^{th} vertex.

If at any point j we find that the size of the matching found M_j is $n^{1-\delta}$ then we stop the process since $|M| \geqslant |M_j| = n^{1-\delta} \geqslant n^{-\delta} \cdot |M^*|$, since M^* cannot have more than n edges.

We now show that if at any point $|M_{i-1}| < n^{1-\delta}$ then $|M_i| = |M_{i-1}| + 1$ with probability at least $1/2$. This is true because the current number of matched vertices is $|V(M_{i-1})| < 2n^{1-\delta}$ and the degree of the i^{th} vertex is at least $4n^{1-\delta}$. So even a single query is enough to find an unmatched neighbor with probability at least $1/2$. If $q < 1$ then we make a query with probability q so we find an unmatched neighbor with probability at least $q/2$. Let $p := \min(1/2, q/2)$ be the probability of finding an unmatched neighbor (this covers

both cases). If we find an unmatched neighbor then we can greedily add that edge to the matching to increase its size by 1.

For $i \in [|M^*|/2]$ let Y_i be the random variable which is 1 if $|M_i| = |M_{i-1}| + 1$ i.e. the i^{th} vertex in the order finds an unmatched neighbor. We have $\mathbb{E}[Y_i] = \Pr(Y_i = 1) \geqslant p$. Let $Y = \sum_i Y_i$ be the size of the final matching $M_{|M^*|/2}$. We have

$$\mathbb{E}[Y] = \sum_i \mathbb{E}[Y_i] \geqslant p \cdot |M^*|/2 \geqslant \min(|M^*|/4, q \cdot n^\delta/4) \geqslant \min(|M^*|/4, 20n^{1-\delta}\log n) = \mu_L.$$

Each Y_i is independent and in $[0,1]$ so we can apply the Chernoff bound (Proposition 2.4.3) with $\varepsilon = 1/2$:

$$\Pr(Y < \mu_L/2) \leqslant \exp(-\mu_L/10) \leqslant \exp(-|M^*|/40) \leqslant \exp(-n^\delta/40) \ll 1/\text{poly}(n).$$

Thus, $|M| \geqslant Y > \mu_L/2 = \min(|M^*|/8, 10n^{1-\delta}\log n) \geqslant n^{-\delta}|M^*|$ with probability $1 - 1/\text{poly}(n)$. Note that this analysis will fail if at some step i we find a matching of size $n^{1-\delta}$. If this happens then we can only guarantee that the size of the matching M is at least $n^{1-\delta}$. Therefore, we get that $|M| \geqslant \min(n^{1-\delta}, n^{-\delta}|M^*|) \geqslant n^{-\delta}|M^*|$. ∎

Proof of Theorem 31. The proof of the approximation (Lemma 10.5.1) follows from Claims 10.5.2 and 10.5.3. When $q \geqslant 1$, the number of queries used in Algorithm 21 is $O(n^{1-2\delta}\log n)$ for each vertex implying $O(n^{2-2\delta}\log n)$ queries in total.

When $q < 1$, we make a query on each vertex independently with probability q. For $i \in [n]$ let Y_i be the random variable which is 1 if we make a query for vertex i. We have $\mathbb{E}[Y_i] = \Pr(Y_i = 1) = q$. Let $Y = \sum_i Y_i$ be the total number of queries made. We have $\mathbb{E}[Y] = \sum_i \mathbb{E}[Y_i] = q \cdot n = 80n^{2-2\delta}\log n = \mu_H$. Each Y_i is independent and in $[0,1]$ so we can apply the Chernoff bound (Proposition 2.4.3) with $\varepsilon = 1$:

$$\Pr(Y > 2\mu_H) \leqslant \exp(-\mu_L/4) \leqslant \exp(-20n^{2-2\delta}\log n) \ll 1/\text{poly}(n).$$

Thus, the number of queries is at most $2\mu_H = O(n^{2-2\delta}\log n)$ with probability $1 - 1/\text{poly}(n)$. ∎

References

[AAD+23] Vikrant Ashvinkumar, Sepehr Assadi, Chengyuan Deng, Jie Gao, and Chen Wang. Evaluating Stability in Massive Social Networks: Efficient Streaming Algorithms for Structural Balance. In Nicole Megow and Adam Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 132, 139, 140, 142, 148

[AB21] Sepehr Assadi and Soheil Behnezhad. Beating Two-Thirds For Random-Order Streaming Matching. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 35, 162

[ABB+19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635. SIAM, 2019. 35

[ABB+05] Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Vizing's theorem in near-linear time. 205. 1

[ABK+24] Sepehr Assadi, Soheil Behnezhad, Christian Konrad, Kheeran K. Naidu, and Janani Sundaresan. Settling the pass complexity of approximate matchings in dynamic graph streams. *CoRR*, abs/2407.21005, 2024. 292

[ABKL23] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 131–144, New York, NY, USA, 2023. Association for Computing Machinery. 6, 12, 292

[Abl96] Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theoretical Computer Science*, 175(2):139–159, 1996. 25

[ABM23] Antonios Antoniadis, Hajo Broersma, and Yang Meng. Online graph coloring with predictions. *arXiv preprint arXiv:2312.00601*, 2023. 13, 204, 205

[ACG+15] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2237–2246. JMLR.org, 2015. 93, 128, 129, 131

[ACK+16] Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 311–319, 2016. 5, 8, 32, 33, 36, 44

[ACK19a] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for (Δ + 1) vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019. 56, 71

[ACK19b] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. *Sublinear Algorithms for ($\Delta + 1$) Vertex Coloring*, pages 767–786. 2019. 4, 11, 287

[ACL+22] Sepehr Assadi, Vaggos Chatziafratis, Jakub Lacki, Vahab Mirrokni, and Chen Wang. Hierarchical clustering in graph streams: Single-pass algorithms and space lower bounds. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume

178 of *Proceedings of Machine Learning Research*, pages 4643–4702. PMLR, 2022. 128

[ACN05] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 684–693. ACM, 2005. 127

[ACN+23] Anders Aamand, Justin Y. Chen, Huy Lê Nguyen, Sandeep Silwal, and Ali Vakilian. Improved frequency estimation algorithms with and without predictions. *CoRR*, abs/2312.07535, 2023. 13, 204

[AD21] Sepehr Assadi and Aditi Dudeja. *A Simple Semi-Streaming Algorithm for Global Minimum Cuts*, pages 172–180. Society for Industrial and Applied Mathematics, 01 2021. 8, 34

[ADRV16] Amirali Abdullah, Samira Daruki, Chitradeep Dutta Roy, and Suresh Venkatasubramanian. Streaming verification of graph properties. In *Proc. 27th International Symposium on Algorithms and Computation*, pages 3:1–3:14, 2016. 228, 229

[ADW+25] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2039, 2025. 162, 168

[AEKM20] Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Fair correlation clustering. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 4195–4205. PMLR, 2020. 128

[AG09] Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 328–338. Springer, 2009. 5, 8, 32, 33

[AG18] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Trans. Parallel Comput.*, 4(4):17:1–17:40, January 2018. 292

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. *Analyzing Graph Structure via Linear Measurements*, pages 459–467. 2012. 3, 5, 7, 10, 55, 56, 57, 58, 71, 72, 227, 228, 236, 246, 253, 257, 273, 282

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, PODS '12, pages 5–14, New York, NY, USA, 2012. Association for Computing Machinery. 5, 7, 37, 55, 71

[AHLW16] Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New characterizations in turnstile streams with applications. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 20:1–20:22, 2016. 72

[AJPS23] Sepehr Assadi, Nirmit Joshi, Milind Prabhu, and Vihan Shah. Generalizing Greenwald-Khanna Streaming Quantile Summaries for Weighted Inputs. In Floris Geerts and Brecht Vandevoort, editors, *26th International Conference on Database Theory (ICDT 2023)*, volume 255 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 4

[AKK18] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *arXiv preprint arXiv:1811.07780*, 2018. 12

[AKK19] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 11

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742. SIAM, 2017. 71, 152

[AKL+22] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 884–895. IEEE, 2022. 1

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364, 2016. 6, 9, 56, 71, 72, 73, 74, 93, 292

[AKM22] Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for δ-coloring. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 234–247, New York, NY, USA, 2022. Association for Computing Machinery. 4

[AKP25] Sepehr Assadi, Sanjeev Khanna, and Aaron Putterman. Correlation clustering and (de) sparsification: Graph sketches can match classical algorithms. *arXiv preprint arXiv:2504.04258*, 2025. 6, 10, 128

[ALMM08] Vladimir E. Alekseev, Vadim V. Lozin, Dmitriy S. Malyshev, and Martin Milanic. The maximum independent set problem in planar graphs. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2008. 203

[Alo97] Noga Alon. On the edge-expansion of graphs. *Combinatorics, Probability and Computing*, 6(2):145–152, 1997. 32

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 20–29, New York, NY, USA, 1996. Association for Computing Machinery. 2, 4

[AN22] Sepehr Assadi and Hoai-An Nguyen. Asymptotically Optimal Bounds for Estimating H-Index in Sublinear Time with Applications to Subgraph

Counting. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*, volume 245 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 11

[ANPW13] Alexandr Andoni, Huy L. Nguyên, Yury Polyanskiy, and Yihong Wu. Tight lower bound for linear sketches of moments. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 25–32. Springer, 2013. 72

[APR13] James Abello, Panos M Pardalos, and Mauricio GC Resende. *Handbook of massive data sets*, volume 4. Springer, 2013. 11

[APT24] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Discrete-smoothness in online algorithms with predictions. *Advances in Neural Information Processing Systems*, 36, 2024. 204

[AR20] Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 342–353. IEEE, 2020. 5

[ARW12] Diogo V Andrade, Mauricio GC Resende, and Renato F Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18:525–547, 2012. 203

[AS19] Sepehr Assadi and Shay Solomon. When Algorithms for Maximal Independent Set and Maximal Matching Run in Sublinear Time. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 11

[AS22] Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages

9:1–9:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. iv, 10, 56, 292

[AS23a] Sepehr Assadi and Vihan Shah. Tight bounds for vertex connectivity in dynamic streams. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 213–227. SIAM, 2023. iv, 9, 227, 229, 236, 257, 265, 266, 268, 280

[AS23b] Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 909–932. IEEE, 2023. 5

[AS25] Sepehr Assadi and Vihan Shah. An improved fully dynamic algorithm for counting 4-cycles in general graphs using fast matrix multiplication. 2025. iv, 12, 17, 292

[ASS20] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 134–144. PMLR, 2020. 13, 204

[Ass24] Sepehr Assadi. *A Simple (1 - ε)-Approximation Semi-Streaming Algorithm for Maximum (Weighted) Matching*, pages 337–354. 2024. 292

[ASW23] Sepehr Assadi, Vihan Shah, and Chen Wang. Streaming algorithms and lower bounds for estimating correlation clustering cost. *Advances in Neural Information Processing Systems*, 36:75201–75213, 2023. iv, 11

[AW08] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. In *Proc. 40th Annual ACM Symposium on the Theory of Computing*, pages 731–740, 2008. 221, 228

[AW22] Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 6, 11, 128, 130

[AY25] Sepehr Assadi and Helia Yazdanyar. Simple sublinear algorithms for $(\delta+1)$ vertex coloring via asymmetric palette sparsification. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 1–8. SIAM, 2025. 4, 11

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. 12, 163, 167

[bas] 4

[Bas08] Surender Baswana. Streaming algorithm for graph spanners? single pass and constant processing time per edge. *Information Processing Letters*, 106(3):110–114, 2008. 4

[BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004. 127

[BBCG10] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3), October 2010. 167

[BBE+22] Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Nearly optimal communication and query complexity of bipartite matching. *arXiv preprint arXiv:2208.02526. To appear in FOCS 2022*, 2022. 56

[BCG20] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 11:1–11:21, 2020. 71

[BCGL23] Siddhartha Banerjee, Vincent Cohen-Addad, Anupam Gupta, and Zhouzi Li. Graph searching with predictions. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 12:1–12:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. 13, 204

[BCMT22] Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual*

Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022, pages 720–731. IEEE, 2022. 6, 128

[BCMT23] Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Single-pass streaming algorithms for correlation clustering. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023 (to appear)*, 2023. 6, 10, 128

[BDD+82] Michael Becker, W Degenhardt, Jürgen Doenhardt, Stefan Hertel, Gerd Kaninke, W Keber, Kurt Mehlhorn, Stefan Näher, Hans Rohnert, and Thomas Winter. A probabilistic algorithm for vertex connectivity of graphs. *Information Processing Letters*, 15(3):135–136, 1982. 54

[BDM+20] Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 517–528. IEEE, 2020. 28

[BDP+25] Vladimir Braverman, Prathamesh Dharangutte, Shreyas Pai, Vihan Shah, and Chen Wang. Fully dynamic adversarially robust correlation clustering in polylogarithmic update time. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025. 6, 128

[BDSW24] Vladimir Braverman, Prathamesh Dharangutte, Vihan Shah, and Chen Wang. Learning-Augmented Maximum Independent Set. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024)*, volume 317 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. iv, 18

[Beh22] Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 873–884, 2022. 11, 19, 287, 288

[Beh23] Soheil Behnezhad. *Dynamic Algorithms for Maximum Matching Size*, pages 129–162. 2023. 12, 292

[BEPvR12] Nicolas Bourgeois, Bruno Escoffier, Vangelis Th. Paschos, and Johan M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012. 203

[Ber14] Bertinoro workshop 2014, problem 64. http://sublinear.info/index.php?title=Open_Problems:64, 2014. Accessed: 2021-09-10. 71

[Ber24] Aaron Bernstein. Improved bounds for matching in random-order streams. *Theory of Computing Systems*, 68(4):758–772, 2024. 34, 35, 51

[BFF+01] Tugkan Batu, Eldar Fischer, Lance Fortnow, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 442–451. IEEE, 2001. 11

[BFL+06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, page 253–262, New York, NY, USA, 2006. Association for Computing Machinery. 4, 167

[BFNP24] Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3534–3557. SIAM, 2024. 13, 204

[BFS86] László Babai, Péter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 337–347, 1986. 220, 221, 227, 239

[BG24] Soheil Behnezhad and Alma Ghafari. Fully dynamic matching and ordered ruzsa-szemerédi graphs. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 314–327, 2024. 12, 292

[BGK13] Francesco Bonchi, David García-Soriano, and Konstantin Kutzkov. Local correlation clustering. *CoRR*, abs/1312.5105, 2013. 6, 128, 129

[BGS15] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in o(log n) update time. *SIAM Journal on Computing*, 44(1):88–113, 2015. 12, 292

[BGTZ23] Eric Balkanski, Vasilis Gkatzelis, Xizhi Tan, and Cherlin Zhu. Online mechanism design with predictions. *CoRR*, abs/2310.02879, 2023. 13, 204

[BH92] Ravi Boppana and Magnús M Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32(2):180–196, 1992. 17, 203

[BHHS11] Boaz Barak, Moritz Hardt, Thomas Holenstein, and David Steurer. Subsampling mathematical relaxations and average-case complexity. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 512–531. SIAM, 2011. 132, 137, 138

[BHNT15] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182, 2015. 71

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996. 32

[BK15] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. 32

[BK22] Soheil Behnezhad and Sanjeev Khanna. *New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS*, pages 3529–3566. 2022. 12, 292

[BKS23a] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\varepsilon)$-approximate matching size in truly sublinear update time. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1563–1588, 2023. 12, 19, 287, 292

[BKS23b] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\epsilon)$-approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 254–266, New York, NY, USA, 2023. Association for Computing Machinery. 19, 287

[BKSV14] Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o(n^(1-2/k)) bits. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pages 531–544. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. 72

[BKSW24] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. *J. ACM*, 71(5), October 2024. 12, 292

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. 5(5):454–465, January 2012. 4

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 73–83, 1990. 11

[BMS20] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *Advances in Neural Information Processing Systems*, 33:20083–20094, 2020. 204

[BNWN25] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. *Commun. ACM*, 68(2):87–94, January 2025. 1

[Bop87] Ravi B Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 280–285. IEEE, 1987. 203

[BOS22] Tomer Berg, Or Ordentlich, and Ofer Shayevitz. On the memory complexity of uniformity testing. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume 178 of *Proceedings of Machine Learning Research*, pages 3506–3523. PMLR, 2022. 128

[BOV13] Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I 40*, pages 244–254. Springer, 2013. 4

[BQS+23] Lorenzo Brusca, Lars C. P. M. Quaedvlieg, Stratis Skoulakis, Grigorios Chrysos, and Volkan Cevher. Maximum independent set: Self-training through dynamic programming. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. 13, 204

[Bra] Jan van den Brand. Complexity term balancer. www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication. 390

[BRR23] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 267–280, New York, NY, USA, 2023. Association for Computing Machinery. 11, 19, 287, 291

[BRR24] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Approximating maximum matching requires almost quadratic time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 444–454, New York, NY, USA, 2024. Association for Computing Machinery. 287

[BRRS23] Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. *Beating Greedy Matching in Sublinear Time*, pages 3900–3945. 2023. 19, 287

[BS80] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980. 28

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I 42*, pages 167–179. Springer, 2015. 12

[BS16] Aaron Bernstein and Cliff Stein. *Faster Fully Dynamic Matchings with Small Approximation Ratios*, pages 692–711. 2016. 12, 292

[BS23] Eric Blais and Cameron Seth. Testing Graph Properties with the Container Method . In *2023 IEEE 64th Annual Symposium on Foundations*

of Computer Science (FOCS), pages 1787–1795, Los Alamitos, CA, USA, November 2023. IEEE Computer Society. 288

[BSHR03] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3cnf properties are hard to test. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 345–354, New York, NY, USA, 2003. Association for Computing Machinery. 288

[BSS12] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012. 32, 35

[BV24] Cristian Boldrin and Fabio Vandin. Fast and accurate triangle counting in graph streams using predictions. In *2024 IEEE International Conference on Data Mining (ICDM)*, pages 31–40, 2024. 15

[BY22] Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing - Problems and Techniques*. Springer, 2022. 11

[BYJKS04] Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. 68

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002. 4

[CALLN23] Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering, 2023. 127

[CALMP24] Vincent Cohen-Addad, Silvio Lattanzi, Andreas Maggiori, and Nikos Parotsidis. Dynamic correlation clustering in sublinear update time. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. 6, 128

[CC09] Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 892–901. SIAM, 2009. 203

[CCAL+24] Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. Understanding the cluster linear program for correlation clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1605–1616, New York, NY, USA, 2024. Association for Computing Machinery. 127

[CCAL+25a] Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, David Rasmussen Lolck, Alantha Newman, Mikkel Thorup, Lukas Vogl, Shuyi Yan, and Hanwen Zhang. Static to dynamic correlation clustering. *arXiv preprint arXiv:2504.12060*, 2025. 6, 128

[CCAL+25b] Nairen Cao, Vincent Cohen-Addad, Shi Li, Euiwoong Lee, David Rasmussen Lolck, Alantha Newman, Mikkel Thorup, Lukas Vogl, Shuyi Yan, and Hanwen Zhang. Solving the correlation cluster lp in sublinear time. *arXiv preprint arXiv:2503.20883*, 2025. 127

[CCE+15] Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Brief announcement: New streaming algorithms for parameterized maximal matching & beyond. In Guy E. Blelloch and Kunal Agrawal, editors, *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 56–58. ACM, 2015. 71

[CCE+16] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344. SIAM, 2016. 6, 9, 72, 73, 74, 93, 292

[CCGT14] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–706, 2014. 221, 225, 228, 229, 230, 231

[CCHM15] Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015, pages 1234–1251. SIAM, 2015. 71

[CCM09] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Annotations in data streams. In *Proc. 36th International Colloquium on Automata, Languages and Programming*, pages 222–234, 2009. 220, 221, 223, 243

[CCM10] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Trans. Algorithms*, 6(3):51:1–51:21, 2010. 72

[CCM+15] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and Arthur-Merlin communication. In *Proc. 30th Annual IEEE Conference on Computational Complexity*, pages 217–243, 2015. 229

[CCM16] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Theory of Computing*, 12(10):1–35, 2016. 7, 8, 34, 35

[CCMT14] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Alg.*, 11(1):Article 7, 2014. 14, 15, 220, 223, 228, 229, 230, 245, 246, 250, 251, 252

[CdG+24] Vincent Cohen-Addad, Tommaso d'Orsi, Anupam Gupta, Euiwoong Lee, and Debmalya Panigrahi. Max-cut with ϵ-accurate predictions. *CoRR*, abs/2402.18263, 2024. 13, 204, 205

[CDGH24] Graham Cormode, Marcel Dall'Agnol, Tom Gur, and Chris Hickey. Streaming Zero-Knowledge Proofs. In Rahul Santhanam, editor, *39th Computational Complexity Conference (CCC 2024)*, volume 300 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:66, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 229

[CE16] Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 820–829. IEEE Computer Society, 2016. 203

[CEI+22] Justin Y Chen, Talya Eden, Piotr Indyk, Honghao Lin, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, Tal Wagner, David Woodruff, and Michael Zhang. Triangle and four cycle counting with predictions in graph streams. In *International Conference on Learning Representations*, 2022. 13, 15, 204, 205

[CEK+15] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, August 2015. 2

[CFPS19] Artur Czumaj, Hendrik Fichtenberger, Pan Peng, and Christian Sohler. Testable properties in general graphs and random order streaming. *arXiv preprint arXiv:1905.01644*, 2019. 35

[CG17] Clément L. Canonne and Tom Gur. An Adaptivity Hierarchy Theorem for Property Testing. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference (CCC 2017)*, volume 79 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:25, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 288

[CG19] Amit Chakrabarti and Prantar Ghosh. Streaming verification of graph computations via graph structure. In *Proc. 33rd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 70:1–70:20, 2019. 228

[CGMV20] Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1786–1802. SIAM, 2020. 35

[CGP+18] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 361–372. IEEE Computer Society, 2018. 8, 33, 36, 38, 41

[CGT20] Amit Chakrabarti, Prantar Ghosh, and Justin Thaler. Streaming Verification for Graph Problems: Optimal Tradeoffs and Nonlinear Sketches.

In Jarosław Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 221, 228, 246

[CGW03] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 524–533. IEEE Computer Society, 2003. 127, 128

[Che20] Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. *Theory Comput.*, 16:1–50, 2020. 228

[CHEVW24] Keren Censor-Hillel, Tomer Even, and Virginia Vassilevska Williams. Fast Approximate Counting of Cycles. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 167

[CHW22] Nicolas Christianson, Tinashe Handina, and Adam Wierman. Chasing convex bodies and functions with black-box advice. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume 178 of *Proceedings of Machine Learning Research*, pages 867–908. PMLR, 2022. 13, 204

[CKK20] Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 19, 287

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow

in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. 1, 19, 31, 54, 286

[CKL+24] Melanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. A $(3+\varepsilon)$-approximate correlation clustering algorithm in dynamic streams. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2861–2880, 2024. 6, 128

[CKP08] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 377–386. ACM, 2008. 127

[CKST19] Charles Carlson, Alexandra Kolla, Nikhil Srivastava, and Luca Trevisan. Optimal lower bounds for sketching graph cuts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2565–2569. SIAM, 2019. 5, 8, 32, 33, 43

[CKT93] Joseph Cheriyan, Ming-Yang Kao, and Ramakrishna Thurimella. Scan-first search and sparse certificates: an improved parallel algorithm for k-vertex connectivity. *SIAM Journal on Computing*, 22(1):157–174, 1993. 9, 55, 381, 382

[CLM+21] Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. 6, 128

[CLN22] Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with sherali-adams. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 651–661. IEEE, 2022. 127

[CLQ17] Lijie Chen, Jian Li, and Mingda Qiao. Nearly instance optimal sample complexity bounds for top-k arm selection. In Aarti Singh and Xiaojin (Jerry)

Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 101–110. PMLR, 2017. 207

[CLTL15] Wei Cao, Jian Li, Yufei Tao, and Zhize Li. On top-k selection in multi-armed bandits and hidden bipartite graphs. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1036–1044, 2015. 207

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. 4

[CMSY15] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlationclustering on complete and complete k-partite graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 219–228. ACM, 2015. 127

[CMT13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013. 228

[COJT+11] Luciano da Fontoura Costa, Osvaldo N Oliveira Jr, Gonzalo Travieso, Francisco Aparecido Rodrigues, Paulino Ribeiro Villas Boas, Lucas Antiqueira, Matheus Palhares Viana, and Luis Enrique Correa Rocha. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, 60(3):329–412, 2011. 160

[CPSZ21] Jakub Chledowski, Adam Polak, Bartosz Szabucki, and Konrad Tomasz Zolna. Robust learning-augmented caching: An experimental study. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1920–1930. PMLR, 2021. 204

[CRT05] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005. 11

[CSVZ22] Justin Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. In *International Conference on Machine Learning*, pages 3583–3602. PMLR, 2022. 204, 205

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endowment*, 5(1):25–36, 2011. 229

[CW09] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214. ACM, 2009. 72

[DBS21] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Transactions on Parallel Computing (TOPC)*, 8(1), April 2021. 2

[DGL+25] Matthew Ding, Alexandro Garces, Jason Li, Honghao Lin, Jelani Nelson, Vihan Shah, and David P. Woodruff. Space complexity of minimum cut problems in single-pass streams. In *16th Innovations in Theoretical Computer Science Conference, ITCS 2025*, 2025. iv, 8

[Die05] Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173:33, 2005. 382, 384

[DK20] Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 6, 7, 9, 56, 71, 72, 73, 74, 292

[DKPP22] Ilias Diakonikolas, Daniel M. Kane, Ankit Pensia, and Thanasis Pittas. Streaming algorithms for high-dimensional robust statistics. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning,*

ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, volume 162 of *Proceedings of Machine Learning Research*, pages 5061–5117. PMLR, 2022. 128

[DLS+16] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In Andrew V. Goldberg and Alexander S. Kulikov, editors, *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, volume 9685 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2016. 203

[DLT23] Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '23, page 99–111, New York, NY, USA, 2023. Association for Computing Machinery. 167

[DP09] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009. 26

[DP14] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1–23, 2014. 19, 286

[DPV25] Yinhao Dong, Pan Peng, and Ali Vakilian. Learning-Augmented Streaming Algorithms for Approximating MAX-CUT. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:24, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 15

[DR96] Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996. 26

[DV13] Abhik Kumar Das and Sriram Vishwanath. On finite alphabet compressive sensing. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 5890–5894. IEEE, 2013. 75

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965. 1, 286

[EF03] Dotan Emanuel and Amos Fiat. Correlation clustering - minimizing disagreements on arbitrary weighted graphs. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2003. 128

[EFS+22] Jon C. Ergun, Zhili Feng, Sandeep Silwal, David P. Woodruff, and Samson Zhou. Learning-augmented k-means clustering. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 204

[EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, September 1997. 9, 12, 55, 381, 382

[EGST12] David Eppstein, Michael T. Goodrich, Darren Strash, and Lowell Trott. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theoretical Computer Science*, 447:44–52, 2012. Combinational Algorithms and Applications (COCOA 2010). 167

[Elk11] Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms (TALG)*, 7(2):1–17, 2011. 4

[ELRS17] Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017. 11, 12

[EMM02] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. PAC bounds for multi-armed bandit and markov decision processes. In Jyrki Kivinen and Robert H. Sloan, editors, *Computational Learning Theory, 15th Annual Conference on Computational Learning Theory, COLT 2002, Sydney, Australia, July 8-10, 2002, Proceedings*, volume 2375 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2002. 207, 208

[EMM06] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *J. Mach. Learn. Res.*, 7:1079–1105, 2006. 207, 208

[ES09] David Eppstein and Emma S Spiro. The h-index of a graph and its application to dynamic subgraph statistics. In *Workshop on Algorithms and Data Structures*, pages 278–289. Springer, 2009. 167

[ET75] Shimon Even and R Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975. 54

[EZ04] Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1+, \varepsilon, \beta)$-spanners in the distributed and streaming models. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 160–168, 2004. 4

[Fei04a] Uriel Feige. Approximating maximum clique by removing subgraphs. *SIAM Journal on Discrete Mathematics*, 18(2):219–225, 2004. 17, 203

[Fei04b] Uriel Feige. On sums of independent random variables with unbounded variance, and estimating the average degree in a graph. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 594–603, 2004. 11

[FGG+23] Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhoň. *Local Distributed Rounding: Generalized to MIS, Matching, Set Cover, and Beyond*, pages 4409–4447. 2023. 292

[FGK06] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: a simple $o(2^{0.288n})$ independent set algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 18–25. ACM Press, 2006. 203

[FHM+20] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785. SIAM, 2020. 35

[Fis04a] Eldar Fischer. The art of uninformed decisions: A primer to property testing. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century Vol 1: Algorithms and Complexity Vol 2: Formal Models and Semantics*, pages 229–263. World Scientific, 2004. 11

[Fis04b] Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004. 288

[Fis20] Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33(3):279–291, 2020. 292

[FKM+04] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2004. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). 3, 4, 5, 55, 71, 253, 292, 381

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *SODA*, volume 5, pages 745–754, 2005. 4

[FKM+08] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 12 2008. 47

[FKN21] Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1894–1913. SIAM, 2021. 56

[FMU21] Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic (1+epsilon)-approximate maximum matching with poly(1/epsilon) passes in the semi-streaming model and beyond. *arXiv preprint arXiv:2106.04179*, 2021. 5

[FMU22] Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with poly$(1/\varepsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 248–260, New York, NY, USA, 2022. Association for Computing Machinery. 292

[Fre85] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. 12

[FS21] Willem Feijen and Guido Schäfer. Using machine learning predictions to speed-up dijkstra's shortest path algorithm. *CoRR*, pages 1–28, 2021. 204, 205

[FS24] Moran Feldman and Ariel Szarf. Maximum matching sans maximal matching: A new approach for finding maximum matchings in the data stream model. *Algorithmica*, 86(4):1173–1209, 2024. 5

[Gav21] Dmitry Gavinsky. The layer complexity of arthur-merlin-like communication. *Theory Comput.*, 17:1–28, 2021. 221, 222

[GG06] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. *Theory Comput.*, 2(13):249–266, 2006. 128, 137

[GG23] Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed mis and approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1777–1790, New York, NY, USA, 2023. Association for Computing Machinery. 292

[GGR98] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998. 11

[GHN+23] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–86. SIAM, 2023. 12

[Gho20] Prantar Ghosh. New verification schemes for frequency-based functions on data streams. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 22:1–22:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 228

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 58–66, 2001. 4

[GKM+22] Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A 3-approximation algorithm for maximum independent set of rectangles. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 894–905. SIAM, 2022. 203

[GKMS19] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019. 35

[GKS05] Martin Grohe, Christoph Koch, and Nicole Schweikardt. The complexity of querying external memory and streaming data. In *Fundamentals of Computation Theory: 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005. Proceedings 15*, pages 1–16. Springer, 2005. 2

[GM09] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009. 35

[GMM24] Suprovat Ghoshal, Konstantin Makarychev, and Yury Makarychev. Constraint satisfaction problems with advice. *arXiv preprint arXiv:2403.02212*, 2024. 204, 205

[GMT15] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 241–247, 2015. 5, 9, 55, 56, 57, 58, 62, 228, 280, 382

[GMT23] Francesco Gullo, Domenico Mandaglio, and Andrea Tagarelli. A combinatorial multi-armed bandit approach to correlation clustering. *Data Min. Knowl. Discov.*, 37(4):1630–1691, 2023. 205

[GO16] Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76:654–683, 2016. 5

[Gol99] Oded Goldreich. Combinatorial property testing (a survey). *Randomization Methods in Algorithm Design*, 43:45–59, 1999. 11

[Gol17] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017. 11

[GP13] Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557, 2013. 12, 292

[GR07] Mira Gonen and Dana Ron. On the benefits of adaptivity in property testing of dense graphs. In Moses Charikar, Klaus Jansen, Omer Reingold, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 525–539, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 288

[GR08] Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008. 11

[GR11] Oded Goldreich and Dana Ron. Algorithmic aspects of property testing in the dense graphs model. *SIAM Journal on Computing*, 40(2):376–445, 2011. 288

[GR13] Tom Gur and Ran Raz. Arthur–Merlin streaming complexity. In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 528–539, 2013. 229

[GS24] Prantar Ghosh and Vihan Shah. New Lower Bounds in Merlin-Arthur Communication and Graph Streaming Verification. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. iv, 19

[GSdSM24] Shubham Gupta, Peter W. J. Staar, and Christian de Sainte Marie. Clustering items from adaptively collected inconsistent feedback. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pages 604–612. PMLR, 2024. 205

[GT01] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 460–469, 2001. 288

[GT19] Venkatesan Guruswami and Runzhou Tao. Streaming hardness of unique games. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 5:1–5:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 128

[GT24] Mohsen Ghaffari and Anton Trygub. Parallel dynamic maximal matching. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '24, page 427–437, New York, NY, USA, 2024. Association for Computing Machinery. 292

[GW89] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Math. Program.*, 45(1-3):59–96, 1989. 127

[GW22] Oded Goldreich and Avi Wigderson. Non-adaptive vs adaptive queries in the dense graph testing model. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 269–275, 2022. 288

[GWN20] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2562–2574. SIAM, 2020. 12

[Hås96] Johan Håstad. Clique is hard to approximate within $n^{1\text{-epsilon}}$. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 627–636. IEEE Computer Society, 1996. 14, 17, 203

[HHH22] Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. Fully Dynamic Four-Vertex Subgraph Counting. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*, volume 221 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 12, 13, 16, 161, 162, 164, 165, 166

[HHLS10] Bjarni V Halldórsson, Magnús M Halldórsson, Elena Losievskaja, and Mario Szegedy. Streaming algorithms for independent sets. In *Automata, Languages and Programming: 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I 37*, pages 641–652. Springer, 2010. 4

[HHS22] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms–a quick reference guide. *ACM Journal of Experimental Algorithmics*, 27:1–45, 2022. 12, 167

[HIKV19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 13, 204

[HJJ+14] Julien M. Hendrickx, Karl Henrik Johansson, Raphaël M. Jungers, Henrik Sandberg, and Kin Cheong Sou. Efficient computations of a security index for false data attacks in power networks. *IEEE Transactions on Automatic Control*, 59(12):3194–3208, 2014. 31

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. 19, 286

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015. 12, 16, 161

[HKP23] Bingbing Hu, Evangelos Kosinas, and Adam Polak. Connectivity oracles for predictable vertex failures. *arXiv preprint arXiv:2312.08489*, 2023. 204, 205

[HLRW24] Monika Henzinger, Jason Li, Satish Rao, and Di Wang. Deterministic near-linear time minimum cut in weighted graphs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3089–3139. SIAM, 2024. 31

[HLS22] Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In *Proceedings of the 2022 Annual*

ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 459–498. SIAM, 2022. 167

[HLS+23] Monika Henzinger, Andrea Lincoln, Barna Saha, Martin P Seybold, and Christopher Ye. On the complexity of algorithms with predictions for dynamic graph problems. *arXiv preprint arXiv:2307.16771*, 2023. 13, 204

[HNO08] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 489–498. IEEE Computer Society, 2008. 72

[HPM03] Sariel Har-Peled and Soham Mazumdar. Coresets for k-means andk-median clustering and their applications. STOC, 2003. 28

[HRG00] Monika R Henzinger, Satish Rao, and Harold N Gabow. Computing vertex connectivity: new bounds from old techniques. *Journal of Algorithms*, 34(2):222–250, 2000. 54

[HRR98] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998. 2, 4

[HRW20] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020. 31

[HSSW12] Magnús M Halldórsson, Xiaoming Sun, Mario Szegedy, and Chengu Wang. Streaming and communication complexity of clique approximation. In *International Colloquium on Automata, Languages, and Programming*, pages 449–460. Springer, 2012. 4

[Hu24] Xiao Hu. Fast matrix multiplication for query processing. *Proc. ACM Manag. Data*, 2(2), May 2024. 163

[HWK24] Diba Hashemi and Weronika Wrzos-Kaminska. Weighted matching in the random-order streaming and robust communication models. *arXiv preprint arXiv:2408.15434*, 2024. 35

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and*

Combinatorics, pages 710–716, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 167

[JLL+20] Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P Woodruff. Learning-augmented data stream algorithms. *ICLR*, 2020. 15

[Joh75] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975. 167

[JS18] Arun Jambulapati and Aaron Sidford. Efficient õ (n/eps) spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503. SIAM, 2018. 36

[JST11] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. 25

[JW13] T. S. Jayram and David P. Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Trans. Algorithms*, 9(3):26:1–26:17, 2013. 72, 228

[JWZ24] Ce Jin, Virginia Vassilevska Williams, and Renfei Zhou. *Listing 6-Cycles*, pages 19–27. 2024. 12, 167

[Kap13] Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697. SIAM, 2013. 5

[Kap21] Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, SODA '21, pages 1874–1893, USA, 2021. SIAM. 34, 35

[Kar94] David R Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 648–657, 1994. 51

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. 1, 31, 35

[Kar10] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010. 13, 17, 203

[KHFH23] Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. Guaranteeing the $\tilde{O}$(agm/out) runtime for uniform sampling and size estimation over joins. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '23, page 113–125, New York, NY, USA, 2023. Association for Computing Machinery. 167

[KKG21] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45, 2021. 162

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. *Dynamic graph connectivity in polylogarithmic worst case time*, pages 1131–1142. 2013. 12

[KKMO07] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007. 204

[KKP18] John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 556–567. IEEE Computer Society, 2018. 71

[KKS14a] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751. SIAM, 2014. 35

[KKS14b] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating max-cut. In *Proceedings of the Twenty-Sixth*

Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1263–1282. SIAM, 2014. 4

[KKS15] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating MAX-CUT. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, SODA '15, pages 1263–1282, USA, 2015. SIAM. 128

[KKSV17] Michael Kapralov, Sanjeev Khanna, Madhu Sudan, and Ameya Velingker. $(1+\omega(1))$-approximation to max-cut requires linear space. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1703–1722. SIAM, 2017. 4

[KL02] David R. Karger and Matthew S. Levine. Random sampling in residual graphs. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 63–66. ACM, 2002. 32

[Kla03] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *Proc. 18th Annual IEEE Conference on Computational Complexity*, pages 118–134, 2003. 227

[Kle69] Daniel Kleitman. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory*, 16(2):232–233, 1969. 54

[KLM+17] Michael Kapralov, Yin Tat Lee, CN Musco, Christopher Paul Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017. 55, 71, 73, 386

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 591–600, New York, NY, USA, 2010. Association for Computing Machinery. 2

[KMBC24] Yuko Kuroki, Atsushi Miyauchi, Francesco Bonchi, and Wei Chen. Query-efficient correlation clustering with noisy oracle. *CoRR*, abs/2402.01400, 2024. 205

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization,*

and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings, pages 231–242. Springer, 2012. 5, 35

[KMM+20] Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, SODA '20, pages 1814–1833, USA, 2020. Society for Industrial and Applied Mathematics. 71

[KMNFT20] Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. *Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples*, pages 1753–1772. 2020. 19, 287

[KMPV19] John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 119–133. ACM, 2019. 4, 12, 167

[KMSS12] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 598–609, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 4, 167

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, 1997. 245

[KN21] Christian Konrad and Kheeran K Naidu. On two-pass streaming algorithms for maximum bipartite matching. *arXiv preprint arXiv:2107.07841*, 2021. 5

[KN24] Christian Konrad and Kheeran K. Naidu. *An Unconditional Lower Bound for Two-Pass Streaming Algorithms for Maximum Matching Approximation*, pages 2881–2899. 2024. 292

[KNN+20] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Maintaining triangle queries under updates. *ACM Trans. Database Syst.*, 45(3):11:1–11:46, 2020. 12, 13, 16, 160, 161, 162

[KNP+17] Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P. Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486. IEEE Computer Society, 2017. 25, 72

[KNW10a] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1161–1178. SIAM, 2010. 72

[KNW10b] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 41–52. ACM, 2010. 4, 72

[Kon15] Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 840–852, 2015. 6, 9, 56, 71, 72, 74

[Kon18a] Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, pages 74–1. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2018. 35

[Kon18b] Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 74:1–74:16, 2018. 93

[Kon21] Christian Konrad. Frequent elements with witnesses in data streams. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2021*, 2021. 71

[KP13] Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 305–320, 2013. 229

[KP14] Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP)*, volume 8572 of *LNCS*, pages 919–930, 2014. 229

[KP20] John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1223–1236. ACM, 2020. 72

[KR03] Ravi Kumar and Ronitt Rubinfeld. Algorithms column: Sublinear time algorithms. *ACM SIGACT News*, 34(4):57–67, 2003. 11

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. 68

[KS96] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. 38

[KS10] Shivaram Kalyanakrishnan and Peter Stone. Efficient selection of multiple bandit arms: Theory and practice. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 511–518. Omnipress, 2010. 207

[KT17] Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. *arXiv preprint arXiv:1702.02559*, 2017. 5

[KT18] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM (JACM)*, 66(1):1–50, 2018. 31

[KTAS12] Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer, and Peter Stone. PAC subset selection in stochastic multi-armed bandits. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012. 207, 395

[LH08] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, page 915–924, New York, NY, USA, 2008. Association for Computing Machinery. 2

[Li21] Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–395, 2021. 31

[Liu24] Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 228–243, 2024. 12, 292

[LLW88] Nathan Linial, Laszlo Lovasz, and Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988. 54

[LMRX21] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 13, 204

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021. 1, 54, 56

[LNW14] Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 174–183, 2014. 29, 72

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020. 31

[LS22] Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. *arXiv preprint arXiv:2205.03930*, 2022. 57

[LSV23] Silvio Lattanzi, Ola Svensson, and Sergei Vassilvitskii. Speeding up bellman ford via minimum violation permutations. In *International Conference on Machine Learning*, pages 18584–18598. PMLR, 2023. 13, 204

[LW13] Yi Li and David P. Woodruff. A tight lower bound for high frequency moment estimation with small error. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, volume 8096 of *Lecture Notes in Computer Science*, pages 623–638. Springer, 2013. 72

[LW16] Yi Li and David P. Woodruff. On approximating functions of the singular values in a stream. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 726–739. ACM, 2016. 152

[M+89] Colin McDiarmid et al. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989. 26

[Mad10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 245–254. IEEE Computer Society, 2010. 32

[MC23] Konstantin Makarychev and Sayak Chakrabarty. Single-pass pivot algorithm for correlation clustering. keep it simple! In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 6412–6421. Curran Associates, Inc., 2023. 6, 128

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005. 5

[McG14] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, may 2014. 2, 28

[MM09] Avner Magen and Mohammad Moharrami. Robust algorithms for MAX INDEPENDENT SET on minor-free graphs based on the sherali-adams hierarchy. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 258–271. Springer, 2009. 203

[MMPS17] Morteza Monemizadeh, Shan Muthukrishnan, Pan Peng, and Christian Sohler. Testable bounded degree graph properties are random order streamable. *arXiv preprint arXiv:1707.07334*, 2017. 35

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020. 34

[MP80] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980. 34

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 27

[MRL99] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 251–262, 1999. 4, 28

[MS25] Slobodan Mitrović and Wen-Horng Sheu. A framework for boosting matching approximation: parallel, distributed, and dynamic. *arXiv preprint arXiv:2503.01147*, 2025. 292

[MSOI+02] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. 160

[MTVV15] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482, 2015. 55, 71

[Mut05] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. 4

[MV80] Silvio Micali and Vijay V Vazirani. An $o(\sqrt{|V||e|})$ algoithm for finding maximum matching in general graphs. In *21st Annual symposium on foundations of computer science (Sfcs 1980)*, pages 17–27. IEEE, 1980. 19, 286

[MV20] Andrew McGregor and Sofya Vorotnikova. Triangle and four cycle counting in the data stream model. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, PODS'20, pages 445–456, New York, NY, USA, 2020. Association for Computing Machinery. 4, 12, 167

[MV22] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022. 204

[MVL+15] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, Christian Bizer, et al. The graph structure in the web: Analyzed on different aggregation levels. *The Journal of Web Science*, 1(1):33–47, 2015. 2

[MVV16] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 401–411. ACM, 2016. 4, 167

[Nis90] Noam Nisan. Pseudorandom generators for space-bounded computation. In *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pages 204–212, 1990. 245

[NO08] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336, 2008. 19, 287

[NPRR18] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3), March 2018. 16, 160, 167

[NRR14] Hung Q Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *Acm Sigmod Record*, 42(4):5–16, 2014. 12, 16, 160, 167

[NS22] Kheeran K. Naidu and Vihan Shah. Space optimal vertex cover in dynamic streams. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 53:1–53:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. iv, 10, 56

[NY19] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1844–1860, 2019. 9, 71, 72, 73

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. STOC '10, page 457–464, New York, NY, USA, 2010. Association for Computing Machinery. 12, 292

[ORRR12] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. *A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size*, pages 1123–1131. 2012. 287

[Pod73] VD Podderyugin. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern*, 2(136):2, 1973. 54

[PR07] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1–3):183–196, August 2007. 19, 287

[PS18] Pan Peng and Christian Sohler. Estimating graph parameters from random order streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2449–2466. SIAM, 2018. 35

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018. 13, 204

[PSY22] Seth Pettie, Thatchaphol Saranurak, and Longhui Yin. Optimal vertex connectivity oracles. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 151–161, New York, NY, USA, 2022. Association for Computing Machinery. 57

[PSYC21] Thomas Pontoizeau, Florian Sikora, Florian Yger, and Tristan Cazenave. Neural maximum independent set. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases - International Workshops of ECML PKDD 2021, Virtual Event, September 13-17, 2021, Proceedings, Part I*, volume 1524 of *Communications in Computer and Information Science*, pages 223–237. Springer, 2021. 13, 204

[PW11] Eric Price and David P. Woodruff. (1 + eps)-approximate sparse recovery. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 295–304. IEEE Computer Society, 2011. 72

[PW13] Eric Price and David P. Woodruff. Lower bounds for adaptive sparse recovery. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 652–663. SIAM, 2013. 72

[Raz90] Alexander A Razborov. On the distributional complexity of disjointness. In *International Colloquium on Automata, Languages, and Programming*, pages 249–253. Springer, 1990. 68

[Rob01] John M Robson. Finding a maximum independent set in time o (2n/4). Technical report, Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001. 203

[Rou21] Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021. 203

[RPS+21] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021. 160

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996. 11

[RS06] Sofya Raskhodnikova and Adam Smith. A note on adaptivity in testing properties of bounded degree graphs. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 13, pages 1–6. Citeseer, 2006. 288

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *Leibniz International Proceedings in Informatics, LIPIcs*, pages 39:1–39:16, Germany, January 2018. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing. 8, 34, 36, 52, 237, 238, 282, 283

[RTC+14] Thomas Rolland, Murat Taşan, Benoit Charloteaux, Samuel J Pevzner, Quan Zhong, Nidhi Sahni, Song Yi, Irma Lemmens, Celia Fontanillo, Roberto Mosca, et al. A proteome-scale map of the human interactome network. *Cell*, 159(5):1212–1226, 2014. 2

[Rub90] Ronitt A Rubinfeld. *A mathematical theory of self-checking, self-testing and self-correcting programs*. University of California, Berkeley, 1990. 11

[SDE+21] Jessica Shi, Laxman Dhulipala, David Eisenstat, Jakub Lacki, and Vahab S. Mirrokni. Scalable community detection via parallel correlation clustering. *Proc. VLDB Endow.*, 14(11):2305–2313, 2021. 127

[SE24] Karim Abdel Sadek and Marek Elias. Algorithms for caching and mts with reduced number of predictions. *arXiv preprint arXiv:2404.06280*, 2024. 13, 204

[Sha25] Vihan Shah. Sublinear time lower bounds for approximating matching size using non-adaptive queries. Unpublished manuscript, 2025. iv, 20

[She09] Jonah Sherman. Breaking the multicommodity flow barrier for o(vlog n)-approximations to sparsest cut. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 363–372. IEEE Computer Society, 2009. 32

[SJR17] Max Simchowitz, Kevin G. Jamieson, and Benjamin Recht. The simulator: Understanding adaptive sampling in the moderate-confidence regime. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, volume 65 of *Proceedings of Machine Learning Research*, pages 1794–1834. PMLR, 2017. 207, 397

[SK23] Satoshi Sugiura and Fumitaka Kurauchi. Isolation vulnerability analysis in road network: Edge connectivity and critical link sets. *Transportation Research Part D: Transport and Environment*, 119:103768, 2023. 31

[SMS+08] Mukund Seshadri, Sridhar Machiraju, Ashwin Sridharan, Jean Bolot, Christos Faloutsos, and Jure Leskove. Mobile call graphs: beyond power-law and lognormal distributions. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 596–604, New York, NY, USA, 2008. Association for Computing Machinery. 2

[SMS+20] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, 29(2-3):595–618, 2020. 2, 160

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, 1995. 27

[ST81] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122, 1981. 12

[ST11] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. 1

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969. 162

[SW15] Xiaoming Sun and David P Woodruff. Tight bounds for graph problems in insertion streams. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *LIPIcs*, pages 435–448, Dagstuhl, Germany, 2015. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 4, 7, 9, 29, 32, 36, 38, 43, 45, 55, 56, 68, 72, 229, 253, 382

[SW23] Clifford Stein and Hao-Ting Wei. Learning-augmented online packet scheduling with deadlines. *CoRR*, abs/2305.07164, 2023. 13, 204

[Swa04] Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 526–527. SIAM, 2004. 127, 128

[SZ22] Roy Schwartz and Roded Zats. Fair correlation clustering in general graphs. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 37:1–37:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 128

[Tar73] Robert Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, 2(3):211–216, 1973. 167

[Tha16a] Justin Thaler. Data stream verification. In *Encyclopedia of Algorithms*, pages 494–499. Springer Berlin Heidelberg, 2016. 229

[Tha16b] Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *Proc. 43rd International Colloquium on Automata, Languages and Programming*, pages 59:1–59:14, 2016. 14, 15, 221, 225, 226, 228, 231, 252, 253

[Tˇ22] Jakub Tětek. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. In Mikołaj Bojańczyk, Emanuela Merelli, and David P.

Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107:1–107:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 163, 167

[UKBM11] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011. 2

[VDBCK+23] Jan Van Den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. 1, 19, 31, 286

[VGW18] Nate Veldt, David F. Gleich, and Anthony Wirth. A correlation clustering framework for community detection. In Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 439–448. ACM, 2018. 127

[Vit01] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001. 2

[Vor20] Sofya Vorotnikova. Improved 3-pass algorithm for counting 4-cycles in arbitrary order streaming. *arXiv preprint arXiv:2007.13466*, 2020. 4, 12, 167

[VWW25] Virginia Vassilevska Williams and Alek Westover. Listing 6-Cycles in Sparse Graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92:1–92:21, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 12, 167

[VY11] Elad Verbin and Wei Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 11–25. SIAM, 2011. 152, 153

[Waj17] David Wajc. Negative association: definition, properties, and applications. *Manuscript, available from https://goo. gl/j2ekqM*, 7, 2017. 51

[WB20] Jose L. Walteros and Austin Buchanan. Why is maximum clique often easy in practice? *Oper. Res.*, 68(6):1866–1895, 2020. 203

[WDL+20] Xinjue Wang, Ke Deng, Jianxin Li, Jeffery Xu Yu, Christian S Jensen, and Xiaochun Yang. Efficient targeted influence minimization in big social networks. *World Wide Web*, 23(4):2323–2340, 2020. 31

[Wes01] Douglas Brent West. **Introduction to graph theory**, volume 2. Prentice hall Upper Saddle River, 2001. 246, 381

[XH22] Jinghui Xia and Zengfeng Huang. Optimal clustering with noisy queries via multi-armed bandit. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 24315–24331. PMLR, 2022. 205

[XN13] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2013. 203

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227. IEEE Computer Society, 1977. 299

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979. 238

[Yu21] Huacheng Yu. Tight distributed sketching lower bound for connectivity. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1856–1873. SIAM, 2021. 72

[YYI09] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 225–234, New York, NY, USA, 2009. Association for Computing Machinery. 19, 287

[YZ97] Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997. 167

[Zel11] Mariano Zelke. Intractability of min- and max-cut in streaming graphs. *Information Processing Letters*, 111(3):145–150, 2011. xvii, 5, 7, 31, 34, 35, 45, 46, 47

[Zuc06] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690, 2006. 14, 17, 203

[ZYH14] Chong Zhang, Julian Yarkony, and Fred A. Hamprecht. Cell detection and segmentation using correlation clustering. In Polina Golland, Nobuhiko Hata, Christian Barillot, Joachim Hornegger, and Robert D. Howe, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2014 - 17th International Conference, Boston, MA, USA, September 14-18, 2014, Proceedings, Part I*, volume 8673 of *Lecture Notes in Computer Science*, pages 9–16. Springer, 2014. 127

APPENDICES

Appendix A

Missing details from Chapter 4

A.1 An Insertion-Only Streaming Algorithm

We describe the insertion-only streaming algorithm for k-vertex-connectivity here. This algorithm has been folklore in the literature already since the introduction of graph streaming model in [FKM+04] (but we are not aware of any explicit reference for this result). Several references in the past attribute the algorithm to [EGIN97] but in fact, it appears that [CKT93] also has almost the complete proof which they present as an online algorithm that outputs a certificate of vertex-connectivity (as in Definition 4.1.1)[1]. We provide the algorithm and its analysis in the insertion-only streaming model in this appendix for completeness.

Preliminaries. Before we present the algorithm we mention two important propositions which will be useful in the analysis of the algorithm. The first is Menger's theorem which gives an equivalent definition of k-vertex-connectivity via vertex-disjoint paths.

Proposition A.1.1 (Menger's Theorem; c.f. [Wes01, Theorem 17])**.** *Let G be an undirected graph and s and t be two non-adjacent vertices. Then the size of the minimum vertex cut for s and t is equal to the maximum number of vertex-disjoint paths between s and t. Moreover, a graph is k-vertex-connected if and only if every pair of vertices has at least k vertex-disjoint paths in between.*

The next is Mader's theorem on existence of k-vertex-connected subgraphs on sufficiently dense graphs.

[1] To the best of our knowledge, the first version of [CKT93] is a technical report in 1991 which predates the conference version of [EGIN97] from 1992.

Proposition A.1.2 (Mader's Theorem; c.f. [Die05, Theorem 1.4.3])**.** *For any $k > 1$, if an undirected graph has at least $2k-1$ vertices and at least $(2k-3)(n-k+1)+1$ edges, it contains a k-vertex-connected subgraph.*

Unlike Menger's theorem, there are not many sources that contain a complete proof of Mader's theorem in the above formulation and with the given parameters (despite being a well-known result mentioned in various sources, e.g., with a different formulation in [Die05, Theorem 1.4.3]). Thus, we also present a simple proof of this theorem in Section A.2 for interested readers.

The insertion-only streaming algorithm. We will reprove the following folklore theorem.

Theorem 32 (cf. [CKT93, EGIN97, SW15, GMT15])**.** *There is a deterministic insertion-only streaming algorithm that given an integer $k \geqslant 1$ before the stream and a graph $G = (V, E)$ in the stream, outputs a certificate H of k-vertex-connectivity of G using $O(kn \log n)$ bits of space.*

The algorithm is very simple: when an edge (u, v) arrives in the stream, store the edge if and only if the number of vertex-disjoint paths between u and v is less than k.

Algorithm 22. An insertion-only streaming algorithm for k-vertex connectivity.

Input: A graph $G = (V, E)$ specified in a stream and an integer k specified at the beginning of the stream.

Output: A certificate H for k-vertex connectivity of G.

1. Let $F = \emptyset$. When any edge $e = (u, v)$ arrives, if the maximum number of vertex-disjoint paths between u and v in (V, F) is less than k, update $F \leftarrow F \cup \{e\}$ (otherwise, F remains unchanged).
2. When the stream ends output $H := (V, F)$ as a certificate for k-vertex connectivity of G.

We start by proving the correctness of the algorithm.

Lemma A.1.3. *Subgraph H output by Algorithm 22 is a certificate for k-vertex-connectivity of G.*

Proof. We need to show that H is k-vertex-connected iff G is k-vertex-connected. If H is k-vertex-connected then G is k-vertex-connected since H is a subgraph of G on the same set of vertices.

Suppose now towards a contradiction that G is k-vertex connected, but H is not. This means that there is a vertex cut X of size at most $k-1$ such that S, X, T is a partition of V and there are no edges between S and T. Since G is k-vertex-connected, it has an edge $e = (u, v)$ between S and T. Edge e is not stored in H by Algorithm 22 and so u and v have at least k vertex disjoint paths between them. But this means that deleting X, a set of at most $k-1$ vertices, cannot disconnect S and T, leading to a contradiction. ∎

We now prove the following claim which will be helpful in proving the space bound.

Claim A.1.4. *The certificate H of Algorithm 22 does not contain any subgraph that is $(k+1)$-vertex connected.*

Proof. Assume for contradiction that H contains a subgraph J that is $(k+1)$-vertex connected. Let $e = (u, v)$ be the last edge added to J by Algorithm 22. By Proposition A.1.1, this means u and v have at least $k+1$ vertex-disjoint paths between them in J and thus have at least k vertex disjoint paths between them in $J - \{e\}$. Therefore, when e arrives in the stream, it is not stored since u and v already have k vertex disjoint paths between them in $H - \{e\}$. But this is a contradiction with e being in H. ∎

Finally, we prove that H contains at most $2kn$ edges.

Lemma A.1.5. *The certificate H of Algorithm 22 contains at most $2kn$ edges.*

Proof. If $n < 2k-1$ then H contains at most $n(n-1)/2 \leqslant 2kn$ edges proving the claim. Thus, consider $n \geqslant 2k-1$. If H has more than $2kn$ edges then by Proposition A.1.2 it contains a $(k+1)$-vertex connected subgraph. But H cannot contain any subgraph that is $(k+1)$-vertex connected by Claim A.1.4. ∎

We can now conclude the proof of Theorem 32.

Proof of Theorem 32. Lemma A.1.3 proves that H is a certificate for k-vertex-connectivity of G. Lemma A.1.5 proves that H contains at most $2kn$ edges implying that Algorithm 22 uses $O(kn \log n)$ bits of space. ∎

A.2 Mader's Theorem

We present a self-contained proof of Mader's theorem in this section for the interested reader. Consider the following restatement of the proposition.

Proposition A.1.2 (Mader's Theorem; c.f. [Die05, Theorem 1.4.3])**.** *For any $k > 1$, if an undirected graph has at least $2k-1$ vertices and at least $(2k-3)(n-k+1)+1$ edges, it contains a k-vertex-connected subgraph.*

Proof. We fix a value of k and prove the proposition by induction on n, the number of vertices. Our induction hypothesis is as follows: For any $t \geqslant 2k$, if an undirected graph has $t-1$ vertices and at least $(2k-3)(t-k)+1$ edges then it contains a k-vertex connected subgraph.

Base case: when $t = 2k$.

We have $m \geqslant (2k-3)(k)+1 = 2k^2-3k+1$. A clique on $2k-1$ vertices has $(2k-1)(2k-2)/2 = 2k^2-3k+1$ edges. Thus, the only graph on $2k-1$ vertices that satisfies the edge lower bound is a clique that is k-vertex-connected and thus has subsets that are k-vertex-connected.

Induction step: We assume the hypothesis for integers up to t and prove it for $t+1$, that is if an undirected graph has t vertices and at least $(2k-3)(t-k+1)+1$ edges then it contains a k-vertex-connected subgraph.

Assume towards a contradiction that there is a graph G with t vertices and at least $(2k-3)(t-k+1)+1$ edges which contains no k-vertex connected subgraph. We first show that G has a large minimum degree.

Claim A.2.1. *G has minimum degree $\delta \geqslant 2k-2$.*

Proof. Consider a vertex v with minimum degree δ. Removing v leaves the graph with $t-1 \geqslant 2k-1$ vertices and $m' \geqslant (2k-3)(t-k+1)+1-\delta$ edges. If $m' \geqslant (2k-3)(t-k)+1$ then G contains a k-vertex-connected subgraph by induction; thus, we need to have $\delta \geqslant 2k-2$. ∎

We know that G is not k-vertex-connected which implies there is a vertex cut X with at most $k-1$ vertices which when deleted disconnects G into components S and $T := V - X - S$. By Claim A.2.1, for any vertex $u \in S$, $\deg(u) \geqslant 2k-2$. Moreover, since there are no edges between S and T, any vertex $u \in S$ has neighbors only in X and S. Thus,

since $|X| < k$, we need S to have at least $k-1$ vertices other than u to satisfy the degree requirement of u, which implies $|S| \geqslant k$. By symmetry, we also have $|T| \geqslant k$.

Let G_1 be the induced subgraph of G on $S \cup X$ with n_1 vertices and let G_2 be the induced subgraph of G on $T \cup X$ with n_2 vertices. Both G_1 and G_2 do not contain any k-vertex-connected subgraphs and have at least $2k-1$ vertices, so they have strictly fewer than $(2k-3)(n_1-k+1)+1$ and $(2k-3)(n_2-k+1)+1$ edges, respectively. We now sum the number of edges m_1 of G_1 and m_2 of G_2:

$$\begin{aligned} m_1 + m_2 &\leqslant (2k-3)(n_1-k+1) + (2k-3)(n_2-k+1) \\ &= (2k-3)(n_1+n_2-2k+2) \\ &\leqslant (2k-3)(t-k+1) \\ &< m \qquad\qquad \text{(Since } m \geqslant (2k-3)(t-k+1)+1\text{)} \end{aligned}$$

But we know $m_1+m_2 \geqslant m$ because G_1 and G_2 cover all edges of G (and can even over count some edges, namely, those with both endpoints in X). Thus, we arrive at a contradiction and such a graph G cannot exist. Therefore, we have shown the induction step and proved the proposition. ∎

Appendix B

Missing details from Chapter 5

B.1 Sparse-Neighborhood Recovery via Exhaustive-Search

We give an alternative and simpler sketch for Problem 4 with the caveat that it uses too much randomness to store efficiently[1] and also requires exponential time. Even though this sketch does not work for the purpose of our algorithm, given that it is much simpler than our `SN-Recovery` sketch, we present it here as a warm-up. Formally, we prove the following lemma.

Lemma B.1.1. *There is a linear sketch for Problem 4 that uses sketch and randomness of size, respectively,*

$$s_0 = s_0(n, a, b, c) = O(a \cdot \log c + b \cdot \log n \cdot \log c) \quad \text{and} \quad O(n \cdot s_0) = O(n \cdot (a \cdot \log q + b \cdot \log n \cdot \log q))$$

bits and outputs a wrong answer with probability at most n^{-10}. The algorithm requires exponential time (in parameters a, b and c).

We use the same vector representation of the problem defined in Eq (5.1) in Section 5.2.2. Our approach is essentially to run equality-test from communication complexity on the vector $x = x(G, S)$; at the end of the stream, once we know the set T, we can search over all possible choices for x, given the promises in Problem 4 and return the one that

[1] While one can use the heavy-machinery yet standard PRG ideas to reduce this randomness (see, e.g. [KLM+17]), the use of PRGs will lead to an $O(\log n)$ space overhead that will break the asymptotic optimality of the algorithm.

passes the equality-test. Given that we can bound the number of choices for x, we can limit the number of equality-tests we need to run.

We will be working in the field $\mathbb{F}_q$ throughout this subsection where q is the smallest prime larger than c. In particular, all computations are in $\mathbb{F}_q$. As $q \geqslant c$, recovery of coordinates of x outside T under $\mathbb{F}_q$ is the same as recovery over the integers. The algorithm is as follows.

Algorithm 23. A simple but (somewhat) inefficient algorithm for Problem 4.

Input: A graph $G = (V, E)$ specified via $\mathbf{vec}(E)$ and a set $S \subseteq V$, defining the corresponding vector $x(G, S)$ in Eq (5.1). A set $T \subseteq V$ specified at the *end* of the stream.

Output: The set of neighbors of S outside T, i.e., $N(S) - T$.

Sketching Matrix:

1. Let $s = 2 \cdot \log\left(2^a \cdot q^a \cdot n^b \cdot q^b \cdot n^{10}\right) \cdot \frac{1}{\log q}$.
2. For $i = 1$ to s: compute $z_i = a_i \cdot x$ where a_i is a vector sampled uniformly from $\mathbb{F}_q^n$.

Recovery:

1. Given T, go over all possible vectors $y \in \mathbb{F}_q^n$, that satisfy the promises of Problem 4 for T.
2. For each guess y, check if $a_i \cdot y = z_i$ for all $i \in [s]$. If it is equal for all $i \in [s]$ then go over y and output its non-zero indices (vertices corresponding to the indices) which are not in T.

We now analyze the correctness of the algorithm.

Claim B.1.2. *For any vector $y \neq x$ considered in Algorithm 23 and an iteration $i \in [s]$, we have $a_i \cdot y \neq z_i$ with probability at least $1 - 1/q$.*

Proof. As $y \neq x$, there should be a coordinate $j \in [n]$ where they have different values. The only way for $a_i \cdot y$ to become equal to $a_i \cdot x$ is if $(a_i)_j \cdot (x_j - y_j)$ is equal to $-\sum_{j' \neq j} (a_i)_{j'} \cdot (x_{j'} - y_{j'})$. Since $(x_j - y_j) \neq 0$, there is only one choice of $(a_i)_j$ that can make this equality

happen even conditioned on the rest of a_i. Thus, with probability at least $1 - 1/q$, we have $a_i \cdot y \neq z_i$. ∎

We run s independent iterations thus the failure probability over all iteration is at most $1/q^s$ by Claim B.1.2. We now count the number of possible choices for y in Algorithm 23 given the promises in Problem 4.

Claim B.1.3. *The number of choices for vector y is at most $2 \cdot 2^a \cdot q^a \cdot n^b \cdot q^b$.*

Proof. $N(S)$ contains a subset of T of size at most a, thus there are 2^a possible choices for elements of $N(S)$ within T. $N(S)$ contains at most b elements outside T, thus there are $\sum_{i=0}^{b} \binom{n-a}{i} \leqslant \sum_{i=0}^{b} n^i \leqslant 2n^b$ choices for elements of $N(S)$ outside T. Also, each element can take values between 0 and $q-1$ in x. Thus, the total number of choices for x is at most $2 \cdot 2^a \cdot q^a \cdot n^b \cdot q^b$. ∎

A union bound over all choices for vector y in Claim B.1.3, using the fact that the probability that each one is mistaken for x is only $1/q^s$, implies that the output will be wrong with probability at most

$$2 \cdot 2^a \cdot q^a \cdot n^b \cdot q^b \cdot \frac{1}{q^s} = n^{-10}.$$

This concludes the correctness of the algorithm.

The sketch size is also $O(\log q) = O(\log c)$ bits for each z_i and thus $s \cdot O(\log q) = O(a \log c + b \log n) = s_0$ bits over all as desired. The number of random bits needed however is $O(n \log q)$ in each iteration implying $O(n \cdot (a \log c + b \log n))$ random bits in total. This concludes the proof of Lemma B.1.1.

We again note that the randomness used by this algorithm is too much for our final algorithm to be able to store and on top of that the algorithm requires exponential time for its recovery.

Appendix C

Missing details from Chapter 7

C.1 A Simple Algorithm for Counting 4-Cycles

In this section, we give a very simple algorithm to maintain the total number of 4-cycles in a fully dynamic general graph with worst-case update time $O(n)$ where n is the number of vertices in the graph.

Lemma C.1.1. *There is an algorithm to maintain the total number of 4-cycles in a fully dynamic general graph with worst-case update time $O(n)$ where n is the number of vertices in the graph.*

We will do this by counting the number of 4-cycles through every new edge update. The idea is to maintain the number of wedges i.e. two-paths between every pair of vertices. During an edge update (u, v) we can iterate over all the neighbors of u and then use the number of wedges to v stored in the data structure. We can also update the number of wedges between all pairs of vertices in $O(n)$ worst-case update time. We show this in the following claim.

Claim C.1.2. *The number of wedges between all pairs of vertices can be maintained in $O(n)$ worst-case update time.*

Proof. Let (u, v) be the current edge update. We go over all the neighbors w of u (at most n) and update the number of wedges between w and v. We do the same for v. The total time taken is $O(n)$. ∎

One important thing to note is that we need to be careful about whether we update the data structures first or answer the queries first. The data structures for the wedges should not have the current edge (u, v) as part of their paths. To make this happen we answer the query first during an edge insertion and update the data structures first during an edge deletion. We now show that we can answer the query of the number of 4-cycles through the new edge update in $O(n)$ time.

Claim C.1.3. *Counting the number of 4-cycles through the new edge update takes $O(n)$ worst-case time.*

Proof. Let (u, v) be the current edge update. We iterate over all the neighbors (at most n) w_i of u and then use the number of wedges from w_i to v stored in the data structure. This gives us all the 3-paths from u to v but might include some walks as well. We now show this is not the case.

We know that u, v, w_i are all distinct. Let x be the vertex between w_i and v. x is different from w_i and v because the graph does not have self loops. x is different from u because the edge (u, v) does not exist in the graph. This is because if the update was an edge insertion we answer the query first, so the data structures do not contain the edge (u, v). Otherwise, if the update was an edge deletion we update the data structures first, so they do not contain the edge (u, v). ∎

Claim C.1.2 and Claim C.1.3 together prove Lemma C.1.1.

C.2 Verifying Constraints

In this section, we verify that the solutions we provided for the values of ε and δ for the main algorithm (Section 7.5) and ε_1 and ε_2 for the algorithm where A, C are fixed (Section 7.3), actually satisfy all the constraints. We use the complexity term balancer in [Bra] to get the values of $\omega(\cdot, \cdot, \cdot)$ and find the optimal values of the parameters. We start with the values of ε and δ for the main algorithm (since the values for the algorithm where A, C are fixed, are a function of ε).

Main algorithm with current best bound on the matrix multiplication exponent

In Section 7.4, we claimed that $\varepsilon = 0.0098109$ and $\delta = 3\varepsilon = 0.0294327$ satisfy all the constraints when $\omega = 2.371339$.

- It is easy to verify that Eq (7.11) is satisfied: $\varepsilon \leqslant 1/6$ since $1/6 \geqslant 0.16$.

- It is easy to verify that Eq (7.10) is satisfied: $3\varepsilon \leqslant \delta$ since $\delta = 3\varepsilon$.
- Finally, we need to verify Eq (7.9): $1 - \delta \geqslant (2\omega + 1) \cdot \varepsilon + (\omega - 1)2/3$.

  Substituting $\delta = 3\varepsilon$ we get: $(6\omega + 12)\varepsilon \leqslant 3 - 2(\omega - 1)$.

  The left-hand side of the equation evaluates to 0.2573206187706 and the right-hand side of the equation evaluates to 0.2573220000000003 satisfying the equation. Thus, all the equations are satisfied.

Main algorithm with best possible bound on the matrix multiplication exponent

In Section 7.4, we claimed that $\varepsilon = 1/24$ and $\delta = 3\varepsilon = 1/8$ satisfy all the constraints when $\omega = 2$.

- It is easy to verify that Eq (7.11) is satisfied: $\varepsilon \leqslant 1/6$ since $\varepsilon = 1/24$.
- It is easy to verify that Eq (7.10) is satisfied: $3\varepsilon \leqslant \delta$ since $\delta = 3\varepsilon$.
- Finally, we need to verify Eq (7.9): $1 - \delta \geqslant (2\omega + 1) \cdot \varepsilon + (\omega - 1)2/3$.

  The left-hand side is $7/8$ because $\delta = 1/8$. The right-hand side is $5/24 + 2/3 = 7/8$ since $\omega = 2$. Thus, all the equations are satisfied.

Algorithm where A, C are fixed with current best bounds on the rectangular matrix multiplication exponents

In Section 7.3.4, we claimed that $\varepsilon_1 = 0.04201965$ and $\varepsilon_2 = 0.14568075$ when $\varepsilon = 0.0098109$.

- It is easy to verify that Eq (7.8) is satisfied: $\varepsilon_1 - \varepsilon_2 \leqslant 1/3$ since $\varepsilon_1 - \varepsilon_2 \leqslant 0$.
- It is easy to verify that Eq (7.7) is satisfied: $\varepsilon_1 \leqslant 1/6$ since $1/6 \geqslant 0.16$.
- It is easy to verify that Eq (7.6) is satisfied: $3\varepsilon_1 + 2\varepsilon \leqslant \varepsilon_2$ since $3\varepsilon_1 + 2\varepsilon = \varepsilon_2$.
- Consider Eq (7.5): $\omega(2/3 + 2\varepsilon, 1/3 - \varepsilon_1 + \varepsilon_2, 1/3 - \varepsilon_1 + \varepsilon_2) \leqslant 4/3 - 2\varepsilon_1$.
  The terms inside $\omega(\cdot, \cdot, \cdot)$ on left-hand side are $2/3+2\varepsilon \leqslant 0.6764776$ and $1/3-\varepsilon_1+\varepsilon_2 \leqslant 0.436994434$. We get $\omega(2/3 + 2\varepsilon, 1/3 - \varepsilon_1 + \varepsilon_2, 1/3 - \varepsilon_1 + \varepsilon_2) + 2\varepsilon_1 \leqslant 1.24039952 + 2(0.04201965) = 1.32443882 < 1/3$. Thus, Eq (7.5) is satisfied.
- Finally, consider Eq (7.2): $\omega(1/3 + \varepsilon_1, 2/3 - \varepsilon_1, 1/3 + \varepsilon_1) \leqslant 4/3 - 2\varepsilon_1$.
  The terms inside $\omega(\cdot, \cdot, \cdot)$ on left-hand side are $1/3 + \varepsilon_1 \leqslant 0.375353$ and $2/3 - \varepsilon_1 \leqslant 0.6246471$. We get $\omega(1/3 + \varepsilon_1, 2/3 - \varepsilon_1, 1/3 + \varepsilon_1)+2\varepsilon_1 \leqslant 1.10495201+2(0.04201965) = 1.18899131 < 1/3$. Therefore, all the equations are satisfied.

Algorithm where A, C are fixed with best possible bounds on the rectangular matrix multiplication exponents

In Section 7.3.4, we claimed that $\varepsilon_1 = 1/24$ and $\varepsilon_2 = 5/24 =$ when $\varepsilon = 1/24$.

- It is easy to verify that Eq (7.8) is satisfied: $\varepsilon_1 - \varepsilon_2 \leqslant 1/3$ since $\varepsilon_1 - \varepsilon_2 \leqslant 0$.
- It is easy to verify that Eq (7.7) is satisfied: $\varepsilon_1 \leqslant 1/6$ since $1/6 \geqslant 0.16$.
- It is easy to verify that Eq (7.6) is satisfied: $3\varepsilon_1 + 2\varepsilon \leqslant \varepsilon_2$ since $3\varepsilon_1 + 2\varepsilon = \varepsilon_2$.
- Consider Eq (7.5): $\omega(2/3 + 2\varepsilon, 1/3 - \varepsilon_1 + \varepsilon_2, 1/3 - \varepsilon_1 + \varepsilon_2) \leqslant 4/3 - 2\varepsilon_1$.
  $\omega(2/3 + 2\varepsilon, 1/3 - \varepsilon_1 + \varepsilon_2, 1/3 - \varepsilon_1 + \varepsilon_2) + 2\varepsilon_1 = 1 + 2\varepsilon + \varepsilon_2 - \varepsilon_1 + 2\varepsilon_1 = 1 + 8/24 = 4/3$.
- Finally, consider Eq (7.2): $\omega(1/3 + \varepsilon_1, 2/3 - \varepsilon_1, 1/3 + \varepsilon_1) \leqslant 4/3 - 2\varepsilon_1$.
  $\omega(1/3 + \varepsilon_1, 2/3 - \varepsilon_1, 1/3 + \varepsilon_1) + 2\varepsilon_1 = 1 + 2\varepsilon_1 < 4/3$. Therefore, all the equations are satisfied.

Appendix D

Missing details from Chapter 8

D.1 A Simple Algorithm for an $O(\log n)$-approximation in the Non-persistent Noise Setting

In this section, we give a simple algorithm to get an $O(\log n)$-approximation using the non-persistent noise oracle. The idea is to sample every vertex independently with probability $1/\log n$ and the query the sampled vertices $O(\log n)$ times.

Lemma D.1.1. *There is an algorithm that gives an $O(\log n)$-approximation using the non-persistent noise oracle with $O(n/\varepsilon^2)$ queries with constant probability.*

Proof. We sample every vertex independently with probability $1/\log n$ and add it to the set U. For every vertex $u \in U$, query it $q := \ln n/\varepsilon^2$ times and compute the majority of the queries.

The claim now is that this is enough to know whether $u \in I^*$ for every $u \in U$ with high probability. We prove this claim by applying the Chernoff bound in Proposition 2.4.3. For any vertex $v \in U \cap I^*$, let the random variable $X_v^i = 1$ if the i^{th} query for vertex v is a "yes" and $X_v^i = 0$ otherwise for $i \in [q]$. The proof is the same for $v \in u \setminus I^*$. Observe that $X_v = \sum_i X_v^i$ is the number of "yes" answers returned by $\texttt{ORC}_{G,I^*}(v)$ out of the q queries. Clearly, we have that $\mathbb{E}[X_v] = (1/2 + \varepsilon) \cdot q$, and X_v is a summation of the independent indicator random variables so, we can apply Proposition 2.4.3 to show that

$$\begin{aligned}\Pr\Big(X_v < \frac{q}{2}\Big) &= \Pr(X_v - \mathbb{E}[X_v] \leqslant -\varepsilon \cdot q) \\ &\leqslant \exp\big(-2 \cdot \varepsilon^2 \cdot q\big) && \text{(applying Proposition 2.4.3)}\end{aligned}$$

$$= \exp(-2 \cdot \ln n)$$
$$= n^{-2}.$$

A union bound over all vertices implies that we know whether $u \in I^*$ for every $u \in U$ with high probability. Also, using the Chernoff bound, we can say that with high probability, the number of vertices that are sampled in U is at most $2n/\log n$. Thus, the total number of queries is at most $2n/\varepsilon^2$.

We sample every vertex with probability $1/\log n$ so we sample $|I^*|/\log n$ vertices from I^* in expectation. Firstly, we note that if $|I^*| \leqslant 10 \log n$ the outputting a single vertex gives a $O(\log n)$-approximation. So we consider the case where $|I^*| \geqslant 10 \log n$. For any vertex $v \in V$, let the random variable $Y_v = 1$ if the vertex v is sampled in U and $Y_v = 0$ otherwise. Let $Y_{I^*} = \sum_{v \in I^*} Y_v$ be the random variable denoting the number of elements of I^* sampled in U. Clearly, we have that $\mathbb{E}[Y_v] = 1/\log n$, and Y_{I^*} is a summation of the independent indicator random variables so, we can apply Proposition 2.4.3 with $\delta = 0.5$ to show that:

$$\begin{aligned}
\Pr(Y_{I^*} < \mathbb{E}[Y_{I^*}]/2) &\leqslant \exp(-\mathbb{E}[Y_{I^*}]/10) \\
&= \exp(-|I^*|/10 \log n) \\
&\leqslant \exp(-10 \log n/10 \log n) \qquad \text{(Using the lower bound on } |I^*|) \\
&= 1/e.
\end{aligned}$$

Thus, with constant probability we sample at least $|I^*|/2 \log n$ vertices from I^*. So we are able to recover a $O(1/\log n)$ fraction of I^* giving us an $O(\log n)$-approximation. ∎

Observe that if $|I^*| = \omega(\log n)$ then the above lemma holds with probability $1 - o(1)$. Algorithm 16 improves the approximation factor from $O(\log n)$ to $O(1)$.

D.2 A Lower Bound for the Non-persistent Noise Setting

In this section, we give lower bounds showing that it is not possible to solve the non-persistent noise setting with $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$ using *only* the pure exploration algorithms in MABs. This lower bound shows that to get the desired $O(1)$-approximation with $O(n/\varepsilon^2)$ samples, we must use the fact that the oracle is based on a maximum independent set and utilize structural properties therein.

In what follows, we divide the lower bounds into two results: the first lower bound is for returning the *entire* MIS set I^*, and the second lower bound works with the stronger

setting where we only need to return an $O(1)$ fraction of the vertices in I^*. A caveat here is that for technical reasons, our second lower bound works with arms with *Gaussian* distributions (see for details). A stronger lower bound with Bernoulli distributions is still interesting to pursue.

D.2.1 A lower bound for obtaining the entire maximum independent set

We give a lower bound for getting the entire set I^* of the maximum independent set. The starting point of our lower bound is a top-k pure exploration lower bound by [KTAS12]. The statement of the lower bound is as follows.

Proposition D.2.1 ([KTAS12], rephrased)**.** *Let $\mathcal{D}(n,k)$ be a distribution of Bernoulli MABs instances as follows.*

$\mathcal{D}(n,k)$: a distribution of Bernoulli multi-armed bandits with n arms such that $k \leqslant \frac{n}{2}$.

- In both cases, let the first arm be with mean reward $\frac{1}{2}$.
- Sampling of the high-reward arms:
  1. With probability $1/2$, uniformly at random sample k coordinates $S \subseteq \{2, \cdots, n\}$, and let the arms with mean rewards $\frac{1}{2} + \varepsilon$.
  2. With probability $1/2$, uniformly at random sample $(k-1)$ coordinates $S \subseteq \{2, \cdots, n\}$, and let the arms with mean rewards $\frac{1}{2} + \varepsilon$.
- For the rest of the arms, e.g., arms in coordinates $\{2, \cdots, n\} \setminus S$, let their mean rewards be $\frac{1}{2} - \varepsilon$.

Then, any algorithm that finds the top k arms on $\mathcal{D}(n,k)$ with probability at least $9/10$ has to use $\Omega(\frac{n}{\varepsilon^2} \cdot \log k)$ samples.

We now state our formal lower bound for obtaining the entire set I^* of the maximum independent set, which is a straightforward application of Proposition D.2.1.

Lemma D.2.2. *Any algorithm that given* only *the oracle $\texttt{ORC}_{G,I^*}$ for graph with $|I^*| = \omega(1)$ (but* not *the graph G), returns the entire set of the maximum independent set I^* with probability at least $9/10$ has to use $\omega(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$.*

Proof. Note that to prove the lemma statement, we only need to show that there *exists* a distribution of graphs such that an algorithm that returns the entire set of I^* with q queries will imply an algorithm that finds the top-k arm in $\mathcal{D}(n, k)$ with q queries. Indeed, suppose we are given an algorithm ALG that given access *only* to $\texttt{ORC}_{G,I^*}$, returns the entire set of I^* with probability at least $9/10$ and $O(n)$ queries. We can then construct a hard distribution simply as follows

- With probability $1/2$, let $G = (V, E)$ be with a maximum independent set I^* of size k.
- With probability $1/2$, let $G = (V, E)$ be with a maximum independent set I^* of size $(k-1)$.

By definition, ALG can find the entire set of I^* with probability at least $99/100$ in both cases of the above distribution.

We now construct a distribution of $\mathcal{D}(n', k')$ with n' and k' to be specified later. In the distribution of the MIS instances, we can view the random variable induced by $\texttt{ORC}_{G,I^*}(v)$ for each $v \in V$ as an arm. Furthermore, we can add an extra arm with a mean reward of $1/2$. This means we have constructed a distribution of $\mathcal{D}(n', k)$ with $n' = n + 1$ and $k' = k$ (here, we slightly overload the notation to let n be the number of vertices in G).

Since we know the coordinate of the special arm with mean reward $1/2$, we can simply run ALG with arms on coordinates $\{2, 3, \cdots, n\}$. In the end, since we can recover the *entire* set of I^*, we can output with the following rules:

- If the output of ALG is of size k, output the same set of arms;
- If the output of ALG is of size $k-1$, output the set of returned arms and the special arm with mean reward $1/2$.

Note that conditioning on the success of ALG, the reduction algorithm always succeeds in finding the top-k arms.

Finally, let $k = n/2$, and by using Proposition D.2.1, to output the top-k arms, we need $\Omega(\frac{n'}{\varepsilon} \cdot \log k') = \Omega(\frac{n}{\varepsilon} \cdot \log n)$ arm pulls. As such, ALG in the parameter range of $k = n/2$ must use $\Omega(\frac{n}{\varepsilon} \cdot \log n) = \omega(n)$ queries to $\texttt{ORC}_{G,I^*}$, as desired by Lemma D.2.2. ∎

D.2.2 A lower bound for obtaining an $O(1)$-approximation of the maximum independent set

In what follows, we present a slightly more involved lower bound. The lower bound shows that if the algorithm does not look into G, then even the easier task to obtain a *constant fraction* of the MIS set I^* would require $\omega(n/\varepsilon^2)$ queries.

For technical reasons, we work with *Gaussian arms* for this lemma, for which a stronger *instance-dependent* lower bound was obtained by [SJR17]. A definition of the oracle $\texttt{ORC}_{G,I^*}$ with Gaussian reward distributions is given as follows. We define $\texttt{ORC}_{G,I^*} : V \to \mathbb{R}$ as a function that upon queried on a vertex $v \in I^*$, $\texttt{ORC}_{G,I^*}(v)$ returns a random sample from $\mathcal{N}(\frac{1}{2}+\varepsilon, 1)$. Conversely, upon queried on a vertex $u \notin I^*$, $\texttt{ORC}_{G,I^*}(v)$ returns a random sample from $\mathcal{N}(\frac{1}{2}-\varepsilon, 1)$.

For Gaussian arms, [SJR17] provides a lower bound that can be described as follows.

Proposition D.2.3 ([SJR17], rephrased)**.** *Let $\mathcal{D}(n,k)$ be a distribution of Gaussian multi-armed bandit instances as follows.*

> $\mathcal{D}(n,k)$: a distribution of multi-armed bandits instances with n arms.
>
> - Uniformly at random sample k coordinates, let the mean rewards of the corresponding arms be $\frac{1}{2}+\varepsilon$.
> - For the rest of the $(n-k)$ arms, let the mean rewards of the corresponding arms be $\frac{1}{2}-\varepsilon$.

Then, any algorithm that returns the top k arms from $\mathcal{D}(n,k)$ with probability at least $9/10$ has to use at least $C \cdot \frac{k}{\varepsilon^2} \cdot \log(n-k) + \frac{(n-k)}{\varepsilon^2} \log k$ arm pulls for some absolute constant C.

Note that Proposition D.2.3 only states a very weak version of the main results of [SJR17] – the paper actually proved a much stronger lower bound. Nevertheless, as we will see shortly, the current version is already sufficient to prove a desired lower bound, stated as follows.

Lemma D.2.4. *Any algorithm that returns a set $\hat{I}$ with probability at least $9/10$, such that*

a). $\hat{I} \subseteq I^*$;

b). $\left|\hat{I}\right| \geqslant \frac{2}{3} \cdot |I^*|$;

given only *the oracle* $\texttt{ORC}_{G,I^*}$ *with* Gaussian reward *distributions (but* not *the graph* G*) has to use* $\omega(n/\varepsilon^2)$ *queries to* $\texttt{ORC}_{G,I^*}$ *in the worst case.*

We will prove the lemma by contradiction. Assume there exists a $\frac{2}{3}$-approximation algorithm for the non-persistent noise setting that uses $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$ (let us call it ALG). We show that we can use this algorithm to recover I^*.

Algorithm 24. An algorithm for obtaining MIS I^* using a $\frac{2}{3}$-approximation algorithm for I^*.
Input: A set V of n vertices; access to $\texttt{ORC}_{G,I^*}$; a 2/3-approximate algorithm ALG.
Output: The MIS I^*.
Parameters: $|I^*| = k = \left(1 - \frac{1}{\log n}\right) \cdot n\,, \qquad r := \log_{1.5} \log n\,, \qquad t := 100 \ln \ln n.$

1. Let $I = \emptyset$ and $V_0 = V$.
2. For $i = 1$ to r:
   - Run ALG for t times.
   - Let I_i be the set of vertices that have been selected at least $t/2$ times.
   - Let $V_i \leftarrow V_{i-1} \setminus I_i$ and $I \leftarrow I \cup I_i$.
3. For each $v \in V_r$, query oracle $2 \log n/\varepsilon^2$ times and compute the majority answer for them.
4. Add to I all vertices whose majority answer is "yes" and then return I.

Claim D.2.5. *When the* $2k/3$*-approximation algorithm (*ALG*) is run for* t *times, it is correct* $8t/10$ *times with probability* $1 - 1/\ln^2 n$.

Proof. A single run of the $2k/3$-approximation algorithm is correct with probability $9/10$ and fails with probability $1/10$. Across t independent runs, in expectation it will succeed $9t/10$ times.

Let the random variable $X_i = 1$ if the i^{th} run of the algorithm is correct and $X_i = 0$ otherwise for $i \in [t]$. Observe that $X = \sum_i X_i$ is the number of times the algorithm is correct. We have that $\mathbb{E}[X] = 9t/10$, and X is a summation of the independent indicator random variables so, we can apply Proposition 2.4.3 to show that

$$\Pr(X - \mathbb{E}[X] \leqslant -t/10) \leqslant \exp(-2t/100) \qquad \text{(applying Proposition 2.4.3)}$$

$$= \exp(-2 \cdot \ln \ln n)$$
$$= 1/\ln^2 n.$$

Thus, the $2k/3$-approximation algorithm is correct $8t/10$ times with probability $1 - 1/\ln^2 n$. ∎

We now condition on the event in Claim D.2.5 for the rest of the proof. A vertex which is not in I^* will be reported at most $2t/10 < t/2$ times so it is never be added to I. Since different vertices can be returned as part of 2/3-approximation across different runs, we wish to bound the number of vertices added to I after each round. Let k_i be the remaining size of I^* at the beginning of round i i.e. $k_i = \left|I^* \setminus \cup_{j=1}^{i-1} I_j\right|$.

Claim D.2.6. *At least $k_i/3$ vertices are added to I in round i i.e. $|I_i| \geqslant k_i/3$.*

Proof. We will prove this using a charging argument. For each of the $8t/10$ iterations, we will put a charge of 1 on a vertex that was not output in the 2/3 fraction of I^*. The total charge on a vertex corresponds to the number of times it was not output. If that exceeds $t/2$ then the vertex is not added to I^* by Algorithm 24 in round i.

The total amount of charge we distribute is $8t/10 \cdot k/3$ since only $k/3$ vertices are not output in each of the $8t/10$ correct rounds. This charge is distributed over k vertices so the average charge on a vertex is $8t/30$. The number of vertices that could have a charge of $\geqslant t/2$ is at most $8k/15 < 2k/3$. Thus, at least $k/3$ vertices have a charge of $< t/2$ and are added to I^* in round i. ∎

We now show that after r iterations the remaining size of I^* is $k/\log n$.

Claim D.2.7. *After r iterations the remaining size of the MIS I^* is $k/\log n$.*

Proof. In each round, we recover at least $1/3$ fraction of the remaining vertices of I^* (Claim D.2.6). This means that for all $i \in [r]$ we have:

$$\left|I^* \setminus \cup_{j=1}^{i} I_j\right| \leqslant \frac{2}{3} \cdot \left|I^* \setminus \cup_{j=1}^{i-1} I_j\right|.$$

Therefore we have

$$\left|I^* \setminus \cup_{j=1}^{r} I_j\right| \leqslant \left(\frac{2}{3}\right)^r \cdot |I^*| = k/\log n.$$

∎

Thus the remaining number of MIS vertices are $k/\log n \leqslant n/\log n$ and the total number of non-MIS vertices are $n/\log n$ (by the choice of k). Therefore querying all the remaining vertices $2\log n/\varepsilon^2$ times gives us $4n/\varepsilon^2$ additional queries. As a result, with probability $1 - 1/n^2$ we identify the remaining vertices in I^* and are able to recover it completely. A union bound over all the vertices implies the correctness of this with probability $1 - 1/n$.

Claim D.2.8. *Algorithm 24 uses $O((\log\log n)^2)$ calls to the $\frac{2}{3}$-approximation algorithm and $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$ and returns I^* with probability $1 - o(1)$.*

Proof. It is easy to observe that the output of the algorithm is I^*. This is because we know that we do not add any non-MIS vertices to I and in the last step we add all the remaining vertices of I^* to I implying that the returned value $I = I^*$.

We know that the number of queries in the last step is $4n/\varepsilon^2$. Also, the number of calls to the $\frac{2}{3}$-approximation algorithm is $r \cdot t = O((\log\log n)^2)$.

Finally, we have conditioned on the event in Claim D.2.5 and we run $O((\log\log n)^2)$ independent copies of it. Thus, by a union bound we get a failure probability of at most $O((\log\log n)^2/\log n)$. Finally, we union bound over the failure probability of $1/n$ from the last step giving us a failure probability of $O((\log\log n)^2/\log n) = o(1)$. ∎

We now finalize the proof of Lemma D.2.4.

Proof of Lemma D.2.4. We prove this by contradiction. We assume towards a contradiction that there is an algorithm that returns a set $\hat{I}$ with probability at least $9/10$, such that $\hat{I} \subseteq I^*$ and $\left|\hat{I}\right| \geqslant \frac{2}{3} \cdot |I^*|$ using $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$. We call this the $\frac{2}{3}$-approximation algorithm.

We fix $|I^*| = k = (1 - \frac{1}{\log n}) \cdot n$. We will use this algorithm to recover I^* completely using $o(k/\varepsilon^2 \log(n-k)) = o(n \log n/\varepsilon^2)$ queries.

We know that Algorithm 24 uses $O((\log\log n)^2)$ calls to the $\frac{2}{3}$-approximation algorithm and $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$ and returns I^*. Also, by assumption we know that the $\frac{2}{3}$-approximation algorithm uses $O(n/\varepsilon^2)$ queries to $\texttt{ORC}_{G,I^*}$. Thus, the total number of queries used is $O(n/\varepsilon^2) + O(n/\varepsilon^2) \cdot O((\log\log n)^2) = O(n(\log\log n)^2/\varepsilon^2) = o(n \log n/\varepsilon^2)$. Therefore, we get a contradiction and the lemma holds. ∎

Remark D.2.9. We note that Algorithm 16 also works for the setting when $\texttt{ORC}_{G,I^*}$ returns samples drawn from Gaussian distributions. This is similar to the MAB setting with Gaussian reward distribution. This is possible because we can use Gaussian tail bound in Claim 8.3.1 to get similar concentration results, and the rest of the proof follows.