PhD Thesis

Privatizing & Dynamizing Algorithm Design

A. R. Sricharan

Advised by Monika Henzinger and Gramoz Goranci

Zusammenfassung

Das Entwerfen von Algorithmen entwickelte sich von Ansätzen mit nur minimalen Annahmen über die Einsatzumgebung hin zu solchen, die an eine Vielzahl unterschiedlicher Rahmenbedingungen angepasst sind. Zwei solcher Rahmenbedingungen sind aufgrund praktischer Anforderungen besonders in den Vordergrund gerückt: Zum Einen verlangen Datenschutzgesetze die Privatisierung personenbezogener Daten, was zu einem Interesse an Algorithmen geführt hat, die zusätzlich nachweisbare Datenschutzgarantien erfüllen. Andererseits erfordert die große Menge an Daten, die kontinuierlich erzeugt und gesammelt wird, Algorithmen, die effizient auf solchen dynamischen Daten arbeiten können. In dieser Arbeit untersuchen wir Algorithmen für diese beiden Einsatzgebiete. Ein verbindendes Thema der hier vorgestellten Algorithmen ist die Datenkomprimierung oder -verdichtung, um die betrachteten Probleme zu vereinfachen. Wir zeigen, wie man

- einen annähernd maximalen Fluss, wenn Kanten zum Graphen hinzugefügt werden,
- einen oblivious Routing-Algorithmus mit geringer Überlastung, der in weniger Iterationen benötigt als baumbasierte Router,
- bedingte Härte für klassische Graphenprobleme auf strukturierten Graphenklassen wie Expandern,
- private Präfixsummen eines Vektorstroms mit weniger Fehlern als der Stand der Technik, wenn der Strom spärlich ist,
- private Schätzungen der Anzahl unterschiedlicher Elemente in einer sich entwickelnden Population, und
- einen dichten Teilgraphen privat mithilfe eines privaten Streaming-Präfixsummen-Algorithmus ermittelt,

erhält.

Abstract

Algorithm design has progressed from approaches with only minimal assumptions about the environment that the algorithm operates in, to those that are adapted to work under a variety of settings. Two such settings have come to the forefront due to practical concerns: Data privacy laws require personal data to be privatized, leading to an interest in algorithms that additionally satisfy provable privacy guarantees; at the same time, the large amount of data that is continuously generated and collected necessitates algorithms that can run efficiently on such evolving data. We study algorithms for these two settings in this thesis. A unifying theme of the algorithms presented here is that of data compression or sparsification to simplify the problems considered. We show how to obtain

- an approximately maximum flow as edges get added to the graph,
- a low congestion oblivious routing algorithm that runs in fewer iterations than tree-based routers,
- conditional hardness for classical graph problems on structured graph classes such as expanders,
- private prefix sums of a stream of vectors with smaller error than state-of-the-art when the stream is sparse,
- private estimates of the number of distinct elements in an evolving population, and
- a dense subgraph privately using a private streaming prefix sum algorithm.

Prologue

Acknowledgements

Acknowledgements have a quality which is hard to describe.

They feel like they've been drafted a hundred times in the head of the author, but then put down on the page in a hurry, the clock ticking on their deadline.

Like, they're trying to tell you the most important thing they've ever said - at the very moment the ship is pulling away from the dock.

Tabitha Carvan, *ANU Essays*

I will presently reduce the font size so I can thank more people, and thanks to Vishwa for making me give a talk (+$$), VC for coming, Umang for our IPL debut, Ulli for a painless PhD, Thejaswini for le bakes of le cakes, Thamboo for Sankaranting and the SF↔SJ trips, Teresa for the differential discussions, Tatia for quadrupling my dal consumption, Tammy for being the only effective bonker (other than me ofc), Swami for the collective experience of McDonald's in Italy, Sushant for width reductions, Sunithi for fika, Srivathsan for a smart nostalgia, Sotho for having the same palate as me whose food recommendations I can trust blindly, Sophia for being the most best host and being the least indecisive person in Bao bao and being abstractly German, Shankar for playing mid like a noob and sometimes showing up to ore oru server than, Shadaab for Riga raving with grandmas and babies, Sayan for thesising, Samarth for streaming games I don't play, Samarth for making amazing sambar and laughing at my bhadels, Sagar for the lunches and the teachings, Rudi for a painless PC experience, Roshana for being a hypewoman, Roodabeh for counting things crazily and inspiring me to buy an e-ink display in the future, Rohit for being inexplicably encouraging and effortlessly supportive, Ritwik for being local don of DDPV and knowing everything and divorcing me (still waiting to get divorced once more), Rasmus for thesising, Rajats for angering Halcyon at all the right times (always), Raj for the cooking adventures and the calls, Quanquan for locally densing, Prantar for a chill morning in Hanover, Prajakta for readily agreeing to read the most out-of-the-left-field papers together, pnp for intellectual discussions on Ramesh-width Suresh-free graphs, Philo for a luxurious wine basement, Peter for stories, Nivedita for having unlimited energy, Niklas for the sundry IST conversations, Nidhi for some chill TU lunches, Neha for vibing, Nanoty for knowing things and going to offlane and yes and this is the year I will finally read Pessoa™, Nana for a content-filled weinwanderung, Monika for the research and the advice, Mira for eating foods from the same plates, Max for being terminally online in many of the same ways that I am, Max for a Halloween miracle and being abstractly German, Martin for stories, Martin for stories, Manas for warmth and having a social circle and inviting me to places and the lemon dish that shall not be nameds, Laxman for coauthoring, Lara for taking the tram line that cannot be named that stops at cannot be named and being the only brainrot enjoyer in Bao bao, Kishlaya for his misfortune that resulted in my Italy trip (I shall see you again soon hopefully), Kathrin for the hallway discussions, Karthik for being great company when we were totally not getting killed in Bangalore, Karen for the darndest lunch stories (I believe now that I can join lunches even if I become a prof), Kannan for playing 50000 games without me (we need to meet more), Jyothsna for eating together in all corners of Austria, Jimmy for Zenmaxxing, Jillu for the brekkie talkies and the support, Ice for studying, Hubert for thesising, Harald for understanding problems (how does one do that?), Gramoz for the research and the advice, George for svting, Eva for matching the vibe and bringing the hater mentality to Bao bao (I approve), Dominique for encouraging social event organizing, Dia for very long stories on instagram about set theory I am sorry I could have done a PhD in set theory if I had read them properly and understood what was going on but I couldn't because I didn't spend enough time is it because Instagram is not a place that I go to to be intellectually stimulated maybe idk who knows, Deka for memespamming, David for collaborating on this paper (accept wer), Chittu for the brekkie talkies and the support, Chayan for healing, Charlotte for organizing the next docs retreat (right?), Carol for being surprised there can be someone in computer science who reads books, brim for reducing the time taken to solve hommade crosswords by a couple of hours at the very least, Bardiya for teaching me how to not pronounce Iranian names and Batmaniiii, Bala for saying I'm too funny for my own good (thanks man), Babloo for the hosting and kutti paapa content, Ativ for bingchilling, Aswannth for knowing everything that exists in Tamil and being ore oru movie paakuravan than, Ashwani for being St. Peter the Apostle of Kaiserslautern, Arvind for some one-directional YTP, Arpan for showing Schönbrunn to me even though I'd been here for three years by then, Areeb for being suspiciously free any time I say I come to Innsbrugg and yes and increasing my to-read pile at a leftbooks pace, Aram for repeatedly experimenting how close he can travel to me without having to meet me (no you blaady), AP for Jetlagging at the recommended pace (immediately), Anushka for all our BGA Pandemic losses, Antonis for independently setting max, Antoine for being the lowest key flamer of Bao bao (one day you shall be called out for it I swear) and having petabytes of brain storage for the most random of Taskmaster tidbits, Ankita for many more suuuuunnnnn together and for knowing everything and the discord vs, Anish for pavukontolling together, Anirudh for the wedding runnings, Anamay for calmly gyaan giving or at least having that aura even when not giving any gyaan, Amit for liking lemons unlike *ahem* others, Amik for also being mistaken for a delivery guy and the Chinese dish, Ami for being my first TAAer, Ali for being the best pizza maker in Bao bao and the shui zhu niu, Ahad for valiantly trying to make me not be a couch potato (you failed) and the Manas intro, Aayush for buying my bhadel and promising wholeheartedly to buy all of them even when I plan to make none, and Aaradhya for the custom chess and the punny funs, and straightaway shall I substitute the small script, for some sciency soliloquy.

... если б захотели вполне раздавить, уничтожить человека, наказать его самым ужасным наказанием, так что самый страшный убийца содрогнулся бы от этого наказания и пугался его заранее, то стоило бы только придать работе характер совершенной, полнейшей бесполезности и бессмыслицы.

Федор Достоевский

فزعانة يا قلبي
إكبر بهالغربة
ما تعرفني بلادي
خدني خدني
خدني على بلادي

— فيروز

Bibliographic Note

Part of the contents of this thesis have been published in conference proceedings. Additional bibliographic notes have been provided in the respective chapters when necessary.

- Chapter 4: Gramoz Goranci, Monika Henzinger, Harald Räcke, A. R. Sricharan,
  Incremental Approximate Maximum Flow via Residual Graph Sparsification,
  in *International Colloquium on Automata, Languages, and Programming* (ICALP) 2025.

- Chapter 5: Gramoz Goranci, Monika Henzinger, Harald Räcke, Sushant Sachdeva, A. R. Sricharan,
  Electrical Flows for Polylogarithmic Competitive Oblivious Routing,
  in *Innovations in Theoretical Computer Science* (ITCS) 2024.

- Chapter 6: Monika Henzinger, Ami Paz, A. R. Sricharan,
  Fine-Grained Complexity Lower Bounds for Families of Dynamic Graphs,
  in *European Symposium on Algorithms* (ESA) 2022.

- Chapter 7: Monika Henzinger, A. R. Sricharan, Teresa Anna Steiner,
  Differentially Private Continual Release of Histograms and Related Queries,
  in *Artificial Intelligence and Statistics* (AISTATS) 2025.

- Chapter 8: Monika Henzinger, A. R. Sricharan, Teresa Anna Steiner,
  Private Counting of Distinct Elements in the Turnstile Model and Extensions,
  in *Randomization and Computation* (RANDOM) 2024.

- Chapter 9: Laxman Dhulipala, Monika Henzinger, George Li, Quanquan Liu, A. R. Sricharan, and Leqi Zhu,
  Near-Optimal Differentially Private Graph Algorithms via the Multidimensional AboveThreshold Mechanism,
  in *European Symposium on Algorithms* (ESA) 2025.

1. A Non-Technical Introduction

Here I will show part of what was written with invisible fruit ink:

What, so you don't see anything? That means it really is invisible. If only I could write a whole novel in such ink.

—**Georgi Gospodinov,** *The Physics of Sorrow*

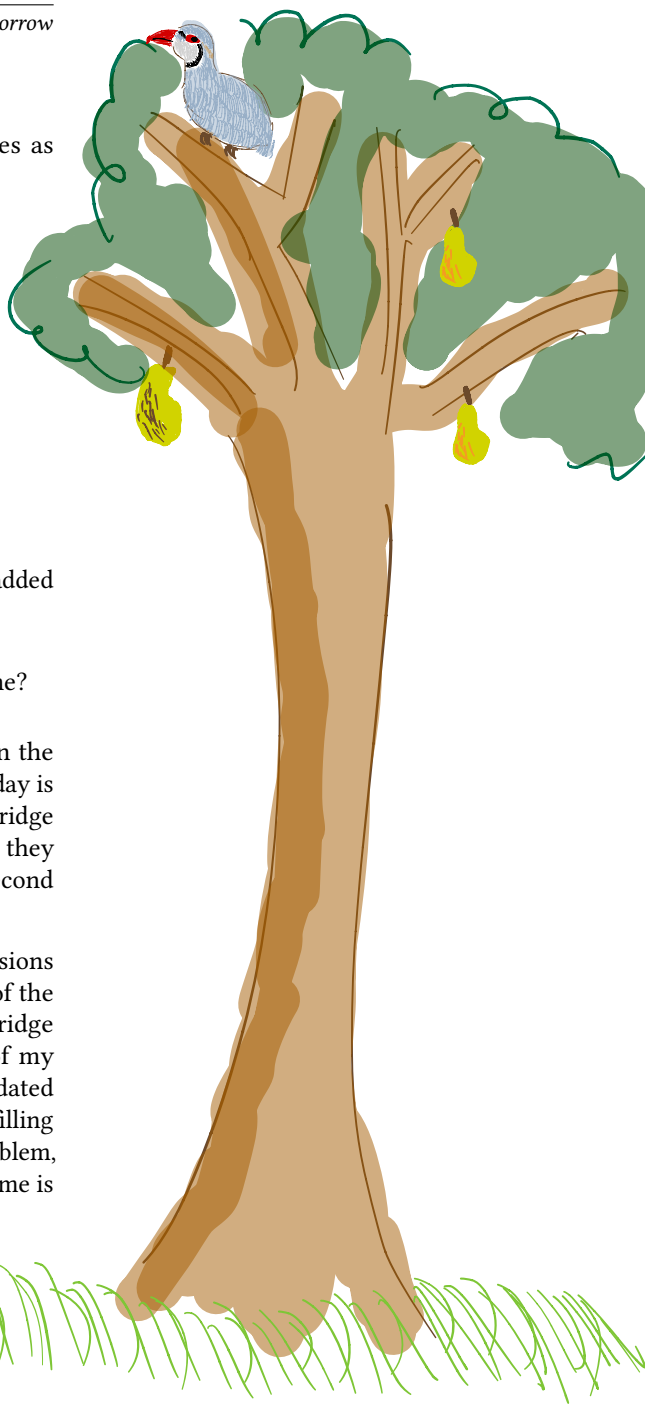


Exhibit A. The nursery rhyme "Twelve Days of Christmas" goes as follows:

> *On the first day of Christmas my true love sent to me*
> *A partridge in a pear tree.*
>
> *On the second day of Christmas my true love sent to me*
> *Two turtle doves,*
> *And a partridge in a pear tree.*
>
> *On the third day of Christmas my true love sent to me*
> *Three French hens,*
> *Two turtle doves,*
> *And a partridge in a pear tree.*
> ⋮

So the rhyme continues, and on the n^{th} day, n copies of a new item are added to the list. Of course, a natural question to ask is the following:

How many total partridges in pear trees did my true love send to me?

If my true love was stingy, they could turn up with no partridges on the second day and say that, really, the partridge they had sent on the first day is what they were talking about on the second day, not a *brand new* partridge in a pear tree (duh). On the other hand, if my true love did not realize they could do this, they would have, to stay true to their message on the second day, simply sent a new partridge each day!

Considering the tax implications of such financially irresponsible decisions by one's partner, it would be my prerogative to inform my true love of the better strategy of hoodwinking me by getting me just a single partridge for this festive season. This sort of concern covers the first half of my thesis, the field of *dynamic algorithms*, where there are continually updated requirements to fulfill, and the question is of how to be stingy while fulfilling these requirements to the letter. This is an important computational problem, as we shall see (while the original problem was to save money, since Time is Money, we convert it to a computational problem of saving time).

Exhibit B. Confidence in their infallibility is how theoretically unbeatable villains meet their match in a puny opponent. Consider the following poem inscribed on the One Ring from Lord of the Rings:

> *One Ring to rule them all, One ring to find them;*
> *One ring to bring them all and in the darkness bind them.*

Sauron, in his hubris, inscribes this onto the One Ring, and it is known across the glens and dales of Middle Earth that the key to his power lies in the One Ring. While such vanity makes for gripping story telling, if Sauron had restricted himself to the humble goal of (Middle) Earth domination, he could have kept the number of rings he made for himself to himself, which would leave the Fellowship in quite a quandary. Is it really worth recruiting a ring-ravaging retinue that rides to Mount Doom if Sauron could have had a secretly hidden Two Ring to replace the One Ring?

Similarly, in a book series that was quite popular at the turn of the 21st century, the evil villain splits his soul into seven soulparts and stores them in various locations that he considers unreachable. To his surprise, the good guy gang find and destroy all of them (even one surprise soulpart that was the good guy all along), then face the remaining part of his soul, the bad guy himself. While there was an extra unintended soulpart that existed in the story, it was unknown to the bad guy himself. Imagine, on the other hand, if he had not only kept the locations but also the *number* of soulparts secret. This would throw a spanner in plan of Goodguy et al. since they would have no way of figuring out if any boss fight was ever "final". The lack of verifiable information reaching the enemy is central to any good strategy, as anyone who has read The Art of War can verify (I haven't).

This question of subverting good storytelling for more strategically complex villains concerns the second half of my thesis, the field of *differential privacy*. Even if someone knew the locations and count of seven of your soulparts, your propaganda should not be helpful for the enemy to figure out if you have an eighth soulpart stashed away somewhere safe. Rewriting the Ring-verse in the language of ε-differential privacy, we would get:

> *One* + $\mathrm{Lap}(1/\varepsilon)$ *Ring(s) to rule them all, One* + $\mathrm{Lap}(1/\varepsilon)$ *ring(s) to find them;*
> *One* + $\mathrm{Lap}(1/\varepsilon)$ *ring(s) to bring them all and in the darkness bind them.*

which quite ruins both the typography and the oomph the original brings, so one could see why Tolkien opted for his version.

How shall I go about solving these problems, you ask.

Inhibit A. You might have heard the following quote, misattributed to very many people:

> *Insanity is doing the same thing over and over again and expecting different results.*

Contrary to this, Chernoff (not directly, but through usage of his bound) insists that you must, absolutely, do that same thing over and over again (about $O(\log n)$ times). He then informs you that you can rest assured that the results of one of your endeavours would be quite good, which would not have been the case had you listened to the above quote. In this thesis, we will take inspiration from Chernoffists and use randomized algorithms to solve problems, sometimes overdoing a few things, just to please Chernoff. One could say that all deterministic algorithms are correct in the same way; every run of a randomized algorithm is wrong in its own way.

Inhibit B. Finally, we take inspiration from this quotation you might have heard growing up if you had your own room.

> *YOUR ROOM LOOKS LIKE A PIGSTY! ARRANGE EVERYTHING AND CLEAN UP YOUR ROOM! RIGHT NOW!*

Sometimes, you keep too many things around that you don't need just because they were given to you. I, for one, have various official-looking letters lying around because they were written in German and I never took the time to run them through a translation app. Maybe I could collect all such letters and organize them together into a neat little pile so its easier for me to get to them when I have the urge to feel productive. I, for two, have two very pretty copies of Catastrophe by Dino Buzzati at home. It was very hard to find, so I would rather have two than none, but I don't really *need* the second one. I could give it away to brighten someone else's day. Maybe you, too, could arrange a few letters neatly and throw some trinkets away.

These timeless concepts have algorithmic counterparts that we can use to hoodwink our partners during Christmas or to offer professional consultations to Sauron in Mordor. Look out for them in the pages that follow.

2. Introduction to the Contents

Why Should You Care About Dynamic Algorithms? Dijkstra, in his seminal paper [1], did not present Dijkstra's algorithm the way it is presented in introductory courses today, with all the bells and whistles. He presented the slow version that runs in $O(n^2)$ time, which does not use fancy data structures to choose the next vertex to explore. This improvement was done by Johnson [2] and Fredman and Tarjan [3] with priority queues, which is a quintessential dynamic problem.

[1]: Dijkstra (1959), "A note on two problems in connexion with graphs"

[2]: Johnson (1972), "On shortest paths and sorting"

[3]: Fredman et al. (1987), "Fibonacci heaps and their uses in improved network optimization algorithms"

Problem 1 (Priority Queue) *Dynamically maintain a collection of pairs of the form (item,priority) with the following operations:*

- INSERT(i, p)*: Insert item i with priority p.*
- EXTRACT_MIN*: Return the item with smallest priority.*
- DECREASE_PRIORITY(i, p')*: Decrease the priority of item i to p'.*

Ford and Fulkerson [4] presented their slow algorithm for finding a maximum flow in a graph. This was improved with link-cut trees [5], a classic dynamic data structure. While these older data structures were graph-independent, the celebrated recent result by Chen, Kyng, Liu, Gutenberg and Sachdeva [6] that computes a maximum flow in almost linear time makes heavy use of dynamic graph algorithms to speed up iterations of static algorithms.

[4]: Ford et al. (1956), "Maximal Flow Through a Network"

[5]: Sleator et al. (1983), "A Data Structure for Dynamic Trees"

[6]: Chen et al. (2022), "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time"

Even if you care only about faster running times for static graph problems, you cannot avoid having to dip your feet into the dynamic world of data structures; they underpin modern algorithm design.

Why Should You Care About Differential Privacy? In the 1990s, the governor of Massachusetts claimed that all patient data released to researchers was anonymized properly. Sweeney, a CS PhD student at the time, showed that this was not true [7] by identifying his medical records from the supposedly "privatized" data and mailing it to him. Over the next few decades, many supposedly privatized data releases for statistical analysis were used to expose embarrassing information about individuals [8, 9].

[7]: Sweeney (1997), "Weaving technology and policy together to maintain confidentiality"

[8]: Narayanan et al. (2008), "Robust Deanonymization of Large Sparse Datasets"

[9]: Tockar (2014 (last accessed 10.2025)), *Riding with the Stars: Passenger Privacy in the NYC Taxicab Dataset*

Sweeney [10] proposed a fix. Coarsen and drop data until every piece of info about you was also shared with k other people. This was widely adopted. Theoretically, however, it was quite untenable. If you happened to share the same shameful private data with $k-1$ other people in a small dataset, then you would leak this information easily; there was no plausible deniability.

[10]: Sweeney (2002), "k-anonymity: A Model for Protecting Privacy"

Dwork, McSherry, Nissim, and Smith [11] introduced a notion of privacy that satisfied a few nice axiomatic properties, and it has been widely adopted, from the US Census [12] to tech giant data collection [13]. It satisfies, among others, the following useful axioms.

[11]: Dwork et al. (2006), "Calibrating Noise to Sensitivity in Private Data Analysis"

[12]: Abowd et al. (2020), "The modernization of statistical disclosure limitation at the US Census Bureau"

[13]: Stojkovic et al. (2022), "Applied Federated Learning: Architectural Design for Robust and Efficient Learning in Privacy Aware Settings"

- Axiom 1: Combine statistics from two privatized datasets, and your guarantee of privacy degrades gracefully (and not catastrophically).
- Axiom 2: Any post-processing of the privatized dataset by someone who cannot access the original dataset does not degrade privacy.

While privacy by itself is easy to guarantee, minimizing the error required to maintain a given level of privacy is an unresolved question for various problems, and is the focus of a long line of fruitful research that has uncovered connections between various fields of theoretical computer science [14].

[14]: Beimel et al. (2022), "Dynamic algorithms against an adaptive adversary: generic constructions and lower bounds"

In this thesis, we study a few important problems in both of these settings. We discuss the specifics of the problems considered and their relevance below before we go into further detail in their respective chapters.

2.1. Graph Algorithms

Routing Without Looking at the Requests

When routing data requests on a network, one would prefer to avoid overloading any single link with too many packets. Usually, this is done by observing where other requests have been routed, then deciding to route the current request along a relatively unused route. Oblivious Routing [15] is a method of routing each request *without having to consider other requests*, while still guaranteeing low congestion on any link of the network. We show in Chapter ElectricalRouting that one can use *electrical flow routing* to reduce the number of iterations required compared to prior algorithms which achieve congestion that is polylogarithmic in the network size.

[15]: Räcke (2002), "Minimizing Congestion in General Networks"

Problems That are Hard on Structured Graphs

A powerful technique that has led to fast algorithms for various classical graph problems is Expander Decomposition [16], a way to decompose the original graph into nice subparts. This works because the problem one wants to solve is easier on an expander graph. We show in Chapter DynamicLBs that some problems are hard even on expanders, which limits the use of this technique as a subroutine for fast algorithms. Other structured graph classes are also considered, such as low maximum degree and power law graphs.

[16]: Saranurak et al. (2019), "Expander Decomposition and Pruning: Faster, Stronger, and Simpler"

Maintaining Flows as Edges Get Added

Ford and Fulkerson [4] have shown how to obtain the maximum possible flow from a source to a target by repeatedly finding source → target paths in a residual graph. Karger and Levine [17] have shown that you can perform Residual Graph Sparsification to get a compact graph where it is easier to find the source → target paths. We show in Chapter IncrementalFlow that one can extend this idea to work even when edges keep getting added to the graph, at the cost of maintaining only a close-to-maximum flow.

[17]: Karger et al. (2015), "Fast Augmenting Paths by Random Sampling from Residual Graphs"

2.2. Differential Privacy

Privately Adding Vectors Lazily

A common problem to solve in machine learning is maintaining a running sum of (high-dimensional) gradients. To privatize these ML systems, you mainly need to maintain a Private Continual Sum of these gradients. When the gradients are single-dimensional and non-negative, Dwork, Naor, Reingold, and Rothblum [18] presented a way of lazily accumulating gradients until they have a significant impact on the sum before performing changes, thereby improving accuracy. In Chapter Histogram, we show a way to extend this lazy accumulation strategy to the high-dimensional setting, and also to the setting where the vector is binary but has negative entries.

[18]: Dwork et al. (2015), "Pure Differential Privacy for Rectangle Queries via Private Partitions"

Privately Counting a Population

As people enter and leave a population, one would like to maintain statistics like, say, the number of different nationalities that belong to the population. Jain, Kalemaj, Raskhodnikova, Sivakumar, and Smith [19] studied this Private Count of Distinct Elements problem where each person is guaranteed that their arrivals and departures over their entire lifetime would be kept private. Their error is parameterized on the maximum number of times a single nationality's count flips from present to absent, and is based on a continual sum mechanism. In Chapter CountDistinct, we give simpler algorithms that are parameterized on the total number of such flips over all nationalities.

[19]: Jain et al. (2023), "Counting Distinct Elements in the Turnstile Model with Differential Privacy under Continual Observation"

Privately Searching for a Dense Region

Detecting a dense region in a banking transaction network indicates fraudulent activity, but users would not want to share their banking data for such statistics in an unprivatized manner. For such settings, Dhulipala, Liu, Raskhodnikova, Shi, Shun, and Yu [20] study the Private Densest Subgraph problem with a stronger notion of privacy where the users privatize their personal data before handing it over. Their accurate algorithms use intricate dynamic data structures as a black-box. In Chapter kCore, we give a clean blackbox reduction from privately finding dense regions to maintaining continual sums privately. Our reduction is simpler and more accurate.

[20]: Dhulipala et al. (2022), "Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs"

Preliminaries 3.

No. This is somewhere to be. This is all you have, but it's still something. Streets and sodium lights. The sky, the world.

ZA/UM, *Disco Elysium*

In this chapter, we discuss some of the high-level preliminaries needed throughout the thesis. For problem-specific preliminaries, we defer them to the chapter they are required for.

3.1. Topic-agnostic Preliminaries

We will denote $\{1, \ldots, n\}$ by $[n]$ throughout the thesis. We will also denote $O(X \log^c(X))$ for any constant $c > 0$ by $\widetilde{O}(X)$, and $O(X^{1+o(1)})$ by $\widehat{O}(X)$.

We use *with high probability* (sometimes abbreviated to whp) to say that the guarantee of a randomized algorithm holds with probability at least $1 - k^{-c}$ for some constant $c > 1$, and a problem-specific parameter k, usually the size of the input. For graph problems, this parameter is n, the number of vertices. For online problems, this parameter is T, the length of the input stream.

3.2. Differential Privacy

Differential privacy is defined for algorithms that take in databases as input, which requires defining a space of all possible databases. We denote such a space by $\mathscr{D}$. The second definition required is that of neighboring databases, which intuitively encodes the concept of two databases, one where your data is present, and one where your data is altered/absent.

Definition 3.2.1 (Databases and Neighbors) *Given a set $\mathscr{D}$ and a symmetric relation $x \sim y$ between pairs of elements $x, y \in \mathscr{D}$, we refer to $\mathscr{D}$ as the* database space *and to elements of $\mathscr{D}$ as* databases. *Two databases x and y are said to be* neighboring *if $x \sim y$ according to the symmetric relation.*

A commonly studied setup is where there is a space of possible database rows $\mathscr{R}$, and each database is made of n database rows for some fixed n. Two databases are said to be neighboring if they differ in at most one row[1].

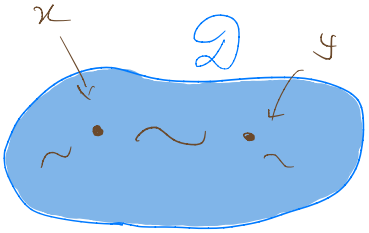
Example 3.2.1 Given a set $\mathscr{R}$ and a natural number $n \in \mathbb{N}$, define

$$\mathscr{D} = \mathscr{R}^n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in \mathscr{R} \ \forall i \in [n]\}.$$

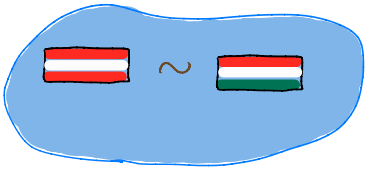
Two databases $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are neighboring if

$$\text{there exists } i \in [n] \text{ such that } x_j = y_j \text{ for all } j \neq i.$$

Differential privacy, introduced by Dwork et al. [11], ensures that the probability of getting the same set of outputs on neighboring databases is close, and this guarantee simultaneously holds for all possible output sets.

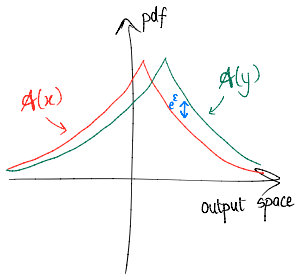


1: It is also possible to consider the setting where n is not fixed, and two databases differ by adding or deleting a database row.



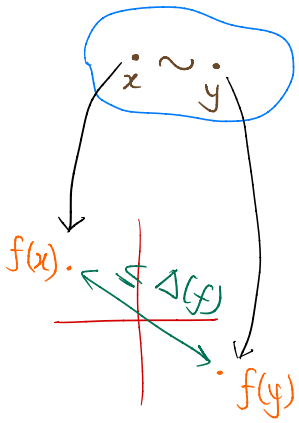
The above figure shows two neighboring databases, Austria and Hungary, that differ only in the last row.

[11]: Dwork et al. (2006), "Calibrating Noise to Sensitivity in Private Data Analysis"



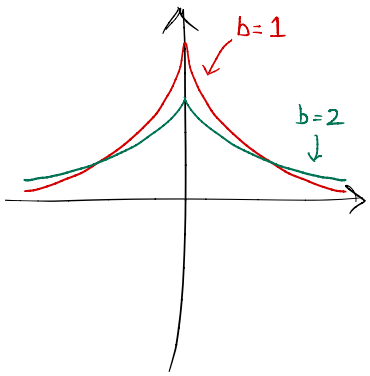
2: This could be fixed by asking for the probability density functions to satisfy the above property instead.

3: On the other hand, they are equivalent for pure dp, and we use this in our proofs.



4: We interchange between Algorithm and Mechanism for algorithms that ensure differential privacy.

[21]: Dwork et al. (2014), "The Algorithmic Foundations of Differential Privacy"



Definition 3.2.2 (Differential privacy) *Given $\varepsilon, \delta \geq 0$, a randomized algorithm $A : \mathcal{D} \to \mathcal{O}$ mapping databases to outputs is* (ε, δ)-differentially private *if for all measurable $S \subseteq \mathcal{O}$ and all neighboring databases $x \sim y$,*

$$\Pr[A(x) \in S] \leq e^{\varepsilon} \cdot \Pr[A(y) \in S] + \delta.$$

If $\delta = 0$ then A is ε-differentially private.

We sometimes abbreviate differential privacy to dp. The general version with $\delta > 0$ is usually referred to as *approximate* differential privacy, and when $\delta = 0$, it is called *pure* differential privacy.

There are two reasons to focus on subsets of outputs and not individual outputs. First, non-discrete distributions could have zero probability for each individual output[2]. Second, the two definitions are not equivalent for approximate differential privacy[3].

For most problems, there is a tradeoff between the privacy and accuracy guarantees that can be ensured under dp. An important property of each problem is its sensitivity, which measures how much the function value can change for neighboring inputs.

Definition 3.2.3 (Sensitivity) *Given a function $f : \mathcal{D} \to \mathbb{R}^k$ defined on databases, the L_p or ℓ_p-sensitivity Δ_p of f is defined as*

$$\Delta_p(f) = \max_{x,y \in \mathcal{D}, x \sim y} \|f(x) - f(y)\|_p.$$

Laplace Mechanism[4]. The canonical distribution that is used to ensure pure differential privacy is the Laplace distribution.

Definition 3.2.4 (Laplace Distribution) *Given $b > 0$, the* Laplace distribution *centered at 0 with scale b is the distribution with density function*

$$f_{\mathrm{Lap}}(b)(x) = \frac{1}{2b} \cdot \exp\left(-\frac{|x|}{b}\right).$$

We use $X \sim \mathrm{Lap}(b)$ or just $\mathrm{Lap}(b)$ to denote a random variable X distributed according to $f_{\mathrm{Lap}}(b)(x)$.

Adding Laplace noise scaled according to the ℓ_1 sensitivity of the function guarantees ε-differential privacy. A proof can be found in Theorem 3.6 of the book by Dwork and Roth [21].

Fact 3.2.1 (Laplace Mechanism) *Let f be any function $f : \mathcal{D} \to \mathbb{R}^k$ with ℓ_1-sensitivity Δ_1. Let $Y_i \sim \mathrm{Lap}(\Delta_1/\varepsilon)$ for $i \in [k]$. The mechanism defined as $A(x) = f(x) + (Y_1, \ldots, Y_k)$ satisfies ε-differential privacy.*

One can then show accuracy guarantees for such mechanisms using the standard Laplace tailbound.

Fact 3.2.2 (Laplace Tailbound) *If $X \sim \mathrm{Lap}(b)$, then $\Pr[|X| \geq t \cdot b] \leq e^{-t}$.*

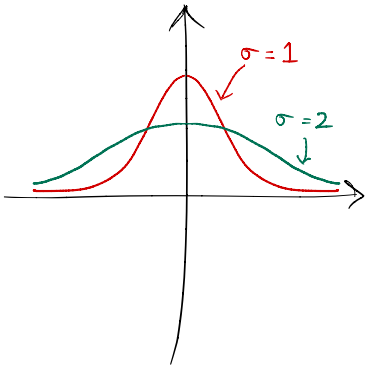
Gaussian Mechanism. To obtain bounds for approximate differential privacy, the canonical mechanism is the Gaussian Mechanism.

Definition 3.2.5 (Gaussian Distribution) *Given $\sigma > 0$, the* Gaussian distribution *centered at 0 with variance σ^2 is the distribution with density function*

$$f_{\mathcal{N}}(\sigma^2)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

We use $X \sim \mathcal{N}(0, \sigma^2)$ or just $\mathcal{N}(0, \sigma^2)$ to denote a random variable X distributed according to $f_{\mathcal{N}}(\sigma^2)(x)$.

This then leads to the following mechanism that depends on the ℓ_2 sensitivity of the function to be privatized. A proof can be found in Theorem A.1 of the book by Dwork and Roth [21].



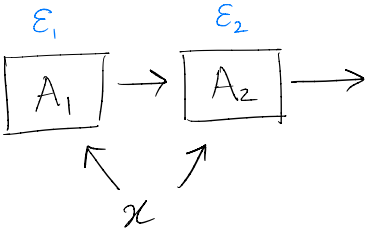
Fact 3.2.3 (Gaussian Mechanism) *Let f be any function $f : \mathcal{D} \to \mathbb{R}^k$ with ℓ_2-sensitivity Δ_2. Let $Y_i \sim \mathcal{N}(0, \sigma^2)$ for $i \in [k]$, where $\sigma \geq \sqrt{2\ln(2/\delta)}\Delta_2/\varepsilon$. The mechanism defined as $A(x) = f(x) + (Y_1, \ldots, Y_k)$ satisfies (ε, δ)-differential privacy.*

Accuracy bounds then follow from the standard Gaussian tailbound.

Fact 3.2.4 (Gaussian tailbound) *If $X \sim \mathcal{N}(0, \sigma^2)$, then*

$$\Pr[|X| \geq \sigma\sqrt{\ln(2/\beta)}] \leq \beta$$

Composition. One can compose two private mechanisms to obtain a new private mechanism with slightly degraded privacy parameters, as shown in Theorems 3.14 and 3.20 of the book by Dwork and Roth [21].



Fact 3.2.5 (Composition Theorem) *Given $\varepsilon_1, \varepsilon_2, \delta_1, \delta_2 \geq 0$, an $(\varepsilon_1, \delta_1)$-differentially private algorithm A_1 with*

$$A_1 : \mathcal{D} \to \text{range}(A_1)$$

and an $(\varepsilon_2, \delta_2)$-differentially private algorithm A_2 with

$$A_2 : \mathcal{D} \times \text{range}(A_1) \to \text{range}(A_2),$$

the mechanism $B : \mathcal{D} \to \text{range}(A_1) \times \text{range}(A_2)$ defined as

$$B(x) = (A_1(x), A_2(x, A_1(x)))$$

is $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-differentially private.

In particular, this gives that composing two pure dp mechanisms gives a pure dp mechanism. The following theorem allows composing multiple mechanisms with a smaller privacy loss in ε, at the cost of a larger δ term.

Fact 3.2.6 (Advanced Composition) *Given $\varepsilon, \delta, \delta' \geq 0$, an (ε, δ)-differentially private algorithm A_1 with*

$$A_1 : \mathcal{D} \to \text{range}(A_1)$$

and (ε, δ)-differentially private algorithms $\{A_i\}_{2 \leq i \leq k}$ with

$$A_i : \mathcal{D} \times \text{range}(A_{i-1}) \to \text{range}(A_i),$$

the composition $B : \mathscr{D} \to \text{range}(A_1) \times \cdots \times \text{range}(A_k)$ defined as

$$B(x) = (A_1(x), A_2(x, A_1(x)), \ldots, A_k(x, A_{k-1}(x)))$$

is $(\varepsilon', k\delta + \delta')$-differentially private, where

$$\varepsilon' = \sqrt{2k \ln(1/\delta')}\varepsilon + k\varepsilon(e^{\varepsilon} - 1).$$

One simple way to obtain an approximate dp mechanism from multiple pure dp mechanisms is by just changing the composition theorem used. This is even the best known asymptotic bound for some problems, like for continual counting!

If we want to achieve a final privacy parameter ε^* when using advanced composition, the following corollary shows how to set the ε values for the intermediate mechanisms.

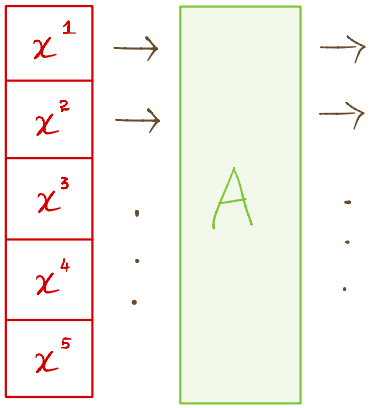
Corollary 3.2.1 (Corollary 3.21 in [21]) *Given $\varepsilon^*, \delta, \delta' \geq 0$ with $\delta', \varepsilon^* < 1$, and mechanisms $A_1, \ldots, A_k$ as in Fact 3.2.6 with*

$$\varepsilon = \frac{\varepsilon^*}{2\sqrt{2k \ln(1/\delta')}},$$

the advanced composition B of $A_1, \ldots, A_k$ is $(\varepsilon^*, k\delta + \delta')$-differentially private.

3.2.1. Continual Observation

Continual observation models mechanisms whose data gets updated regularly, and one needs to output accurate statistics after each update.



Definition 3.2.6 (Continual Observation Stream) *Given a universe of elements $\mathscr{U}$ and a time horizon $T \in \mathbb{N}$, a database in the continual observation model is an input stream $x = x^1, x^2, \ldots, x^T$ with $x^t \in \mathscr{U}$ for all $t \in [T]$.*

Definition 3.2.7 (Continual Observation Mechanism) *Given an input stream $x = x^1, x^2, \ldots, x^T$, a continual observation mechanism A runs for T time steps, and at each time step t, obtains as input $x^t \in \mathscr{U}$ and outputs $A(x^1, \ldots, x^t)$ before proceeding to the next time step. The vector of all T outputs of the mechanism is denoted by $A^T(x)$.*

A common property of a continual mechanism is whether the time horizon T is given as input to the mechanism. A well-studied notion of neighboring databases is one where two neighboring streams differ exactly in one time step, called *event-level* neighboring. This is similar to the row neighboring example mentioned for static databases.

Example 3.2.2 Two streams x and y are event-level neighboring if

$$\text{there exists } t \in [T] \text{ such that } x^j = y^j \text{ for all } j \neq t.$$

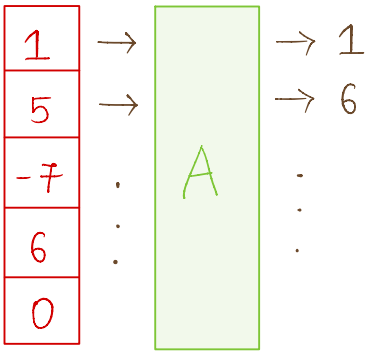
[22]: Dwork et al. (2010), "Differential privacy under continual observation"

Privacy in the continual setting is defined with respect to the outputs of the mechanism over all T time steps, originally defined by Dwork et al. [22].

Definition 3.2.8 (Differential privacy under continual observation) *Given $\varepsilon, \delta \geq 0$, a continual release mechanism $A : \mathscr{U}^T \to \mathscr{O}^T$ is (ε, δ)-differentially private if for all measurable $S \subseteq \mathscr{O}^T$ and for all neighboring streams $x \sim y$*

$$\Pr[A^T(x) \in S] \leq e^{\varepsilon} \cdot \Pr[A^T(y) \in S] + \delta.$$

Continual Counting. A canonical problem in this setting is the continual counting problem, the simplest version of which is to maintain a count of the number of 1s in a stream of bits.



Problem 2 (Continual Counting) *For a universe $\mathcal{U} = \{0, 1\}$ and an input stream $x \in \mathcal{U}^T$, the continual counting problem is to release the prefix sums $a^t = \sum_{i=1}^{t} x^i$ at each time step $t \in [T]$.*

The accuracy measure of problems considered in this thesis is the ℓ_∞ norm of the output vector. In particular, for the continual counting problem, for an algorithm answering $\tilde{a}^1, \ldots, \tilde{a}^T$, the error of the algorithm is

$$\|\tilde{a} - a\|_\infty = \max_{t \in [T]} \{|\tilde{a}^t - a^t|\}$$

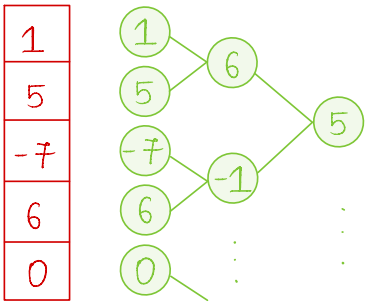
When the universe is non-negative, this is referred to as the *insertions-only* setting. When $\mathcal{U} = \{-1, 0, 1\}$, this is the *turnstile* setting, which allows both insertions and deletions. We also consider the problem where the universe is $\mathbb{Z}$ or $\mathbb{N}$. Here, the event-level neighboring definition given above would allow two streams to differ unboundedly at time t on two neighboring streams. We consider the following restricted model since the unbounded case does not admit accurate algorithms.

Definition 3.2.9 (Continual Counting Neighbors) *For the continual counting problem with $\mathcal{U} = \mathbb{Z}$ or $\mathbb{N}$, two streams x and y are neighboring if*

$$\text{there exists } t \in [T] \text{ such that } x^j = y^j \text{ for all } j \neq t, \text{ and } |x^t - y^t| \leq 1.$$

The asymptotically best algorithm for this problem comes from Dwork et al. [22] and Chan et al. [23], called the *binary tree mechanism.*

[23]: Chan et al. (2011), "Private and Continual Release of Statistics"



Theorem 3.2.2 *Given $\varepsilon > 0$, there is an ε-differentially private mechanism for continual counting that satisfies the following accuracy guarantee: with probability at least $1 - \beta$, at any fixed time step $t \in [T]$, the error is at most*

$$\text{err}_{\text{BT}}(t, \beta, \varepsilon) = O\left(\varepsilon^{-1} \cdot \log t \sqrt{\log 1/\beta} \cdot \max\left\{\sqrt{\log t}, \sqrt{\log 1/\beta}\right\}\right).$$

Taking a union bound over all time steps, with probability at least $1 - \beta$, the additive error is bounded by

$$\text{err}_{\text{BT}}(\beta, \varepsilon) = O\left(\varepsilon^{-1} \cdot \log T \cdot \log(T/\beta)\right).$$

For constant ε and for $\beta = T^{-c}$ for some constant c, this error bound is $O(\log^2 T)$.

Dwork et al. [18] combined this with SVT instantiations that only inserted the count into the binary tree mechanism once it was large enough, to obtain the following mechanism that is optimal on sparse insertions-only streams.

[18]: Dwork et al. (2015), "Pure Differential Privacy for Rectangle Queries via Private Partitions"

Theorem 3.2.3 *Given $\varepsilon > 0$, there is an ε-differentially private insertions-only counting mechanism with the following accuracy guarantee: with probability at least $1 - \beta$, at any fixed time step $t \in [T]$, the error is at most*

$$O\left(\text{err}_{\text{BT}}(\min\{n_t, t\}, \beta, \varepsilon) + \varepsilon^{-1} \cdot \log(t/\beta)\right)$$

where n_t is the true continual count at time t and err_{BT} is the error of the binary tree mechanism.

Continual Histogram. When the input to the continual counting problem is a stream of vectors, it is called the continual histogram problem. Since each vector is a "row" of the database, we refer to each dimension of the vectors as a "column", and the goal is to release all column sums.

Problem 3 (Continual Histogram) *Given a dimension $d \in \mathbb{N}$, a universe $\mathcal{U} = \{0,1\}^d$, and an input stream $x \in \mathcal{U}^T$, the continual histogram problem is to release the prefix column sums $a_j^t = \sum_{i=1}^t x_j^i$ for all $j \in [d]$ at each time step $t \in [T]$.*

The accuracy measure of the histogram problem is an ℓ_∞ norm over all dimensions and all time steps. As in the case of continual counting, when the universe is $\mathbb{Z}$ or $\mathbb{N}$, we consider the restricted neighboring model that also asks that $\|x^t - y^t\|_\infty \leq 1$ at the differing time step t.

A simple histogram mechanism is obtained by composing d single dimensional continual counting mechanisms.

While this theorem composes binary tree mechanisms, one can do this with any continual counting mechanism. In general, an ε-dp counting mechanism with error $\mathrm{err}_{\mathrm{CC}}(t,\beta,\varepsilon)$ when composed gives an ε-dp histogram mechanism with error $\mathrm{err}_{\mathrm{CC}}(t,\beta/d,\varepsilon/d)$. The β is scaled for a union bound, and the ε is scaled for basic composition.

Theorem 3.2.4 *Given $d \in \mathbb{N}$ and $\varepsilon > 0$, there is an ε-differentially private mechanism for continual histogram with the following accuracy guarantee: with probability $\geq 1-\beta$, at any fixed time step $t \in [T]$, the error is at most*

$$\mathrm{err}(d,t,\beta,\varepsilon) = O\left(d\varepsilon^{-1} \cdot \log t \sqrt{\log d/\beta} \cdot \max\left\{\sqrt{\log t}, \sqrt{\log d/\beta}\right\}\right).$$

With probability $\geq 1-\beta$, the additive error over all time steps is at most

$$\mathrm{err}(d,\beta,\varepsilon) = O\left(d\varepsilon^{-1} \cdot \log T \cdot \log(dT/\beta)\right).$$

Adaptive Adversary Model. The current model assumes that the inputs to the counting mechanism do not depend on the previous outputs of the mechanism. We will need to also work with continual counting where the future inputs might depend on past outputs, called the *adaptive adversary* model. We formalize the model and required results in Chapter 7.

[24]: Dwork et al. (2009), "On the complexity of differentially private data release: efficient algorithms and hardness results"

[25]: Lyu et al. (2017), "Understanding the Sparse Vector Technique for Differential Privacy"

3.2.2. The Sparse Vector Technique

The Sparse Vector Technique (often abbreviated as SVT) is based on an algorithm by Dwork et al. [24] and was described more fully by Dwork and Roth [21]. The sparse vector technique is used to answer multiple Yes/No queries while losing privacy linearly only for the Yes answers (and logarithmically for the total number of queries). We present a version in Algorithm 3.1 by Lyu et al. [25] for the special case when $c = 1$. This version allows different thresholds for every query.

Algorithm 3.1: AboveThreshold

Input: Database x, sensitivity bound Δ, thresholds $\{\mathrm{Thresh}_i\}$, and queries $\{q_i\}$ with $\Delta_1(q_i) \leq \Delta$.

$\tau \leftarrow \mathrm{Lap}(2\Delta/\varepsilon)$
for $i = 1, 2, \ldots$ **do**
  $\mu_i \leftarrow \mathrm{Lap}(4\Delta/\varepsilon)$
  if $q_i(D) + \mu_i > \mathrm{Thresh}_i + \tau$ **then**
    output $a_i = \textsc{Yes}$
    Abort
  else
    output $a_i = \textsc{No}$

Lemma 3.2.5 *AboveThreshold is ε-differentially private.*

Lemma 3.2.6 *AboveThreshold fulfills the following accuracy guarantees for $\alpha = 8(\ln k + \ln(2/\beta))/\varepsilon$: for any sequence of queries $q_1, \ldots, q_k$, it holds with probability at least $1-\beta$ that*

- *for an i such that $a_i = \textsc{Yes}$, we have $q_i(D) \geq \mathrm{Thresh}_i - \alpha$, and*
- *for all i such that $a_i = \textsc{No}$, we have $q_i(D) \leq \mathrm{Thresh}_i + \alpha$.*

Part I.

Graph Algorithms

4. Incremental Maximum Flow

Keep only those things that speak to your heart.
Then take the plunge and discard all the rest.
By doing this, you can reset your life and
embark on a new lifestyle.

Marie Kondo, *The Life-Changing Magic of Tidying Up*

4.1. Introduction

The maximum s-t flow problem has been at the forefront of research in theoretical computer science and combinatorial optimization. Algorithms developed for maximum s-t flow and its dual problem, minimum s-t cut, have been highly influential due to their wide applicability [26] and their use as subroutines in other algorithms [27, 28]. A long line of work has improved our understanding of how efficiently maximum flow can be solved in the static setting. Two landmark combinatorial results are the deterministic algorithm by Goldberg and Rao [29], which achieves a running time of $O(\min\{n^{2/3}, m^{1/2}\} \cdot m)$, and the randomized algorithm by Karger and Levine [17] which has a running time of $\widetilde{O}(m + nF^*)$, where F^* is the value of the maximum flow. More recently, efforts building upon a novel blend of continuous optimization techniques, the Laplacian paradigm [30] and graph-based data structures have led to even faster algorithms; notably, a randomized $\widetilde{O}(m/\varepsilon)$ time approximate maximum flow algorithm on undirected graphs [31–34], and a deterministic $\widehat{O}(m)$ time *exact* algorithm for min-cost flow (and thus maximum flow) even on directed graphs [6, 35], which constitutes a major algorithmic breakthrough.

Recently, there has been growing interest in solving the maximum s-t flow problem in the challenging *dynamic* setting, where the goal is to maintain a flow (or its value) under edge insertions and deletions and answer queries about the maintained flow efficiently. For directed graphs, there are strong conditional hardness results [36, 37] showing a lower bound of $\Omega(n)$ and $\Omega(\sqrt{m})$ amortized update time for maintaining *exact* maximum flow, under the OMv conjecture. These strong polynomial lower bounds have motivated a shift in focus toward maintaining *approximate* maximum flows.

Research on maintaining approximate maximum flows in the dynamic setting can be categorized into the following three lines of works. The first line of work [38–40] deals with the fully dynamic setting, supporting both edge insertions and deletions and is based on dynamically maintaining tree-based cut approximations. However, these algorithms require at least a logarithmic loss in the quality of the maintained flow. The second line of work [41–43] is purely combinatorial and works by repeatedly finding augmenting paths in an incremental residual graph together with a lazy rebuilding technique. While these algorithms achieve sub-linear update time, it is not clear how to use them to go beyond the $\sqrt{n}$ barrier on the update time. To address this challenge, the third line of work [35, 40, 44, 45] leverages continuous optimization techniques to maintain a $(1-\varepsilon)$-approximate max flow in the incremental (only edge insertions) and decremental (only edge deletions) settings in $m^{o(1)}$ update time.

[26]: Ahuja et al. (1995), "Applications of network optimization"

[27]: Arora et al. (2012), "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications"

[28]: Sherman (2009), "Breaking the Multicommodity Flow Barrier for O(vlog n)-Approximations to Sparsest Cut"

[29]: Goldberg et al. (1998), "Beyond the Flow Decomposition Barrier"

[17]: Karger et al. (2015), "Fast Augmenting Paths by Random Sampling from Residual Graphs"

[30]: Teng (2010), "The Laplacian Paradigm: Emerging Algorithms for Massive Graphs"

[31]: Sherman (2013), "Nearly Maximum Flows in Nearly Linear Time"

[32]: Kelner et al. (2014), "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations"

[33]: Peng (2016), "Approximate Undirected Maximum Flows in $O(m\text{polylog}(n))$ Time"

[34]: Sherman (2017), "Area-convexity, l_∞ regularization, and undirected multicommodity flow"

[6]: Chen et al. (2022), "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time"

[35]: Chen et al. (2024), "Almost-Linear Time Algorithms for Incremental Graphs: Cycle Detection, SCCs, s-t Shortest Path, and Minimum-Cost Flow"

Table 4.1.: Results on dynamic max flow. The "Flow/Value" column indicates whether the algorithm maintains an actual flow or just the value of a flow. The update time stated is the amortized time over all updates.

Setting	Apx. Factor	Flow/Value	Directed	Weighted	Update Time	Reference
Incremental	1	Flow	Yes	Yes	$O(F^*)$	[41, 42]
	1	Flow	No	No	$\widetilde{O}(n^{2.5}m^{-1})$	[43]
	$1+\varepsilon$	Flow	Yes	No	$\widehat{O}(m^{0.5}\varepsilon^{-0.5})$	[43]
	$1+\varepsilon$	Flow	Yes	Yes	$\widehat{O}(n^{0.5}\varepsilon^{-1})$	[44]
	$1+\varepsilon$	Flow	No	Yes	$O(m^{o(1)}\varepsilon^{-3})$	[45]
	$1+\varepsilon$	Flow	Yes	Yes	$O(m^{o(1)}\varepsilon^{-1})$	[35]
	$1+\varepsilon$	Flow	No	No	$\widetilde{O}(nF^*m^{-1}\varepsilon^{-1})$	Theorem 4.1.1
Decremental	$1+\epsilon$	Value	Yes	Yes	$O(m^{o(1)}\epsilon^{-1})$	[40]
Fully dynamic	$\widetilde{O}(\log n)$	Value	No	Yes	$\widehat{O}(n^{0.667})$	[38]
	$m^{o(1)}$	Value	No	No	$m^{o(1)}$	[39]
	$m^{o(1)}$	Value	No	Yes	$m^{o(1)}$	[40]

[36]: Dahlgaard (2016), "On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter"

[37]: Henzinger et al. (2015), "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture"

[38]: Chen et al. (2020), "Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers"

[39]: Goranci et al. (2021), "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms"

[40]: Brand et al. (2024), "Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality"

[41]: Henzinger (1997), "A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity"

[42]: Gupta et al. (2021), "Simple dynamic algorithms for Maximal Independent Set, Maximum Flow and Maximum Matching"

[43]: Goranci et al. (2023), "Efficient Data Structures for Incremental Exact and Approximate Maximum Flow"

[44]: Brand et al. (2023), "Dynamic Maxflow via Dynamic Interior Point Methods"

[45]: Brand et al. (2024), "Incremental Approximate Maximum Flow on Undirected Graphs in Subpolynomial Update Time"

Focusing on the $(1-\varepsilon)$-approximation regime, the recent partially dynamic maximum flow algorithms suffer from the following two main drawbacks: (i) they depend on the powerful yet intricate machinery of continuous optimization methods, such as interior point methods, monotone multiplicative weight updates, and dynamic min-ratio cycle/cut problems, (ii) all existing algorithms incur an $m^{o(1)}$ factor in the update time, a limitation that arises in many modern dynamic graph-based data structures and appears difficult to overcome. This leads to the following fundamental question:

> Is there an incremental maximum flow algorithm that achieves $(1-\epsilon)$ approximation with polylogarithmic update time?

We answer this question in the affirmative for dense graphs that are undirected and uncapacitated, as summarized in the theorem below.

Theorem 4.1.1 *Given any $\varepsilon \in (0,1)$, there is an incremental randomized algorithm against output-adaptive adversaries that maintains a $(1-\varepsilon)$-approximate maximum s-t flow f under edge insertions on an undirected uncapacitated n-vertex graph G with high probability in total time $O(m\log(n)\alpha(n) + nF^*\log^3(n)/\varepsilon)$, where m is the number of edge insertions, F^* is the value of the max flow after m insertions, and $\alpha(n)$ is the inverse Ackermann function.*

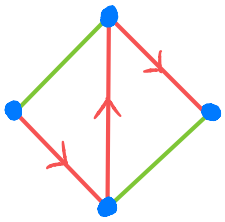
In addition to dense graphs ($m = \Omega(n^2)$), our algorithm achieves polylogarithmic amortized update time even for graphs with small flow value $F^* = \widetilde{O}(m/n)$, regardless of their density. Our algorithm is that it builds upon arguably simple and classic combinatorial techniques, such as residual graph sparsification and cut sparsification.

4.2. Preliminaries

4.2.1. Graphs and Flows

Flows. Our algorithmic results are on undirected uncapacitated graphs $G = (V, E)$. We denote $n = |V|$ and $m = |E|$. For two vertices $s, t \in V$, an s-t flow $f \in \mathbb{R}^m$ assigns a value f_e to each edge such that $\sum_{e\sim v} f_e = 0$ for

all $v \neq s,t$ with $|f_e| \leq 1\ \forall e$. The value of a flow f is $F(f) = \sum_{e \sim s} f_e$, and the maximum flow $f^* = \text{argmax}_{s\text{-}t \text{ flows } f} F(f)$, with value $F^* = F(f^*)$. We use F without the argument f when the flow is clear from context. For any $\alpha \leq 1$, an s-t flow f is an α-approximate flow if $F \geq \alpha \cdot F^*$. If $\alpha > 1$, then the statement holds with $\alpha' = 1/\alpha$.



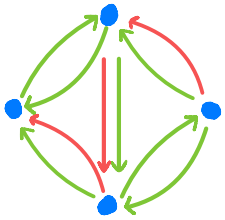
Model. In the incremental model, we start with an empty graph with no edges, and the graph is revealed as a sequence of edge insertions $e_1, e_2, \ldots$, leading to graphs $G_1, G_2, \ldots$. Our goal is to maintain, after each edge insertion e_i, an s-t flow f that is a $(1-\varepsilon)$ approximation to the max s-t flow f_i^* in the current graph G_i. We use F_i^* to denote the value of the flow f_i^*, and we use F^* to denote the final max s-t flow value after all edge insertions.

Directed Graphs. We denote by $\vec{G} = (V, \vec{E})$ a directed (multi-)graph without edge capacities. An edge of integer capacity $x \in \mathbb{Z}^+$ is represented using x parallel edges. For any subset of vertices $S \subseteq V$, $\bar{S} = V \setminus S$ is the complement of S, and $\partial_{\vec{G}}(S)$ denotes the set of edges leaving S, which is also denoted by $\vec{C}(S) = \partial_{\vec{G}}(S)$. Similarly, we use $\overleftarrow{C}(S) = \partial_{\vec{G}}(\bar{S})$ to denote the set of edges entering S. We use $\vec{C}$ and $\overleftarrow{C}$ without the argument S when the subset is clear from context. The capacity $u(\vec{C}) = |\vec{C}|$ of the cut $\vec{C}$ is the number of edges in $\vec{C}$.

While the residual graph G_f of a flow (defined below) is a directed graph, we drop the arrow by convention.

Underlying Undirected Graph. For any directed (multi-)graph $\vec{G}$, we denote by G the underlying undirected (multi-)graph obtained from $\vec{G}$ by forgetting edge orientations, and $\partial_G(S)$ is the set of edges leaving S, which is also denoted by $C(S) = \partial_G(S)$. For a directed cut $\vec{C} = \partial_{\vec{G}}(S)$, we use $C = \partial_G(S)$ to denote the underlying undirected cut in G.

Residual Graph. For any undirected, uncapacitated graph G, the corresponding residual graph for the zero flow $f = 0^m$ is given by G_{0^m} where each edge (u, v) in G is replaced by two directed edges $\vec{uv}$ and $\vec{vu}$. For any G and a non-zero flow f, the residual graph G_f is constructed the following way: start with the graph G_{0^m} as above, and for every edge e that carries flow in the direction $u \to v$, replace the edge $\vec{uv}$ in G_{0^m} with the edge $\vec{vu}$. In that case there are two edges $\vec{vu}$ in G_f. A flow f^* is a maximum flow if and only if G_{f^*} contains no $s \to t$ paths.



4.2.2. Sampling Parameters

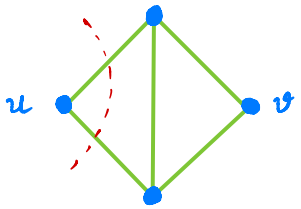
We define our importance parameter for sampling edges next. Since the residual graph has multiple edges between the same vertex pairs, we consider multi-graphs (which allow parallel edges) for the following definition.

Definition 4.2.1 (NI index [46]) *Given an undirected, unweighted (multi-)graph G, an ordered sequence of edge-disjoint spanning forests $T_1, T_2, \ldots$ of is said to be an NI forest packing of G if each T_i is a maximal spanning forest on $G \setminus \cup_{1 \leq j < i} \{T_j\}$, and $\cup_i \{T_i\}$ is a partition of all the edges of G. The NI index ℓ_e of an edge e is the index i of the forest T_i that e belongs to.*

[46]: Nagamochi et al. (1992), "A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph"

While the NI indices are the connectivity parameters that we will use for our sampling scheme due to their resilience to incremental graph updates, the

An example of a graph and it's NI forest packing is given in Section 4.5.



Uniform sampling would involve choosing each edge with equal probability. This is bad for graphs with many small cuts, e.g., a complete graph and a very long path connected by a single vertex.

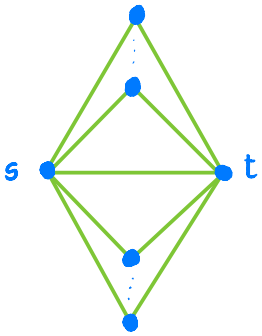
following connectivity parameter is required for our directed sparsification proofs and cut-counting.

Definition 4.2.2 (Edge-connectivity) *Given an undirected unweighted (multi-)graph G and two vertices u and v, the edge connectivity between u and v is defined to be the value of the minimum cut (and thus also the maximum flow) between u and v in G. For an edge $e = (u, v)$, we define the edge-connectivity of e to be the edge connectivity between u and v.*

As we perform *non-uniform* sampling of the edges, we maintain connectivity parameters λ_e that influence the sampling probabilities. The NI index of an edge and the edge-connectivity of an edge are examples of such connectivity parameters. We will show concentration bounds for classes of edges whose λ_es are close to each other, which motivates the following definition.

Definition 4.2.3 (Connectivity class) *Given an undirected graph G and connectivity parameters λ_e for each edge, taking $\Lambda = \max_e \lceil \log \lambda_e \rceil + 1$, the connectivity classes $\mathcal{F} = \{F_i : 1 \le i \le \Lambda\}$ are given by*

$$F_i = \{e : 2^{i-1} \le \lambda_e \le 2^i - 1\}.$$



[47]: Fung et al. (2019), "A General Framework for Graph Sparsification"

For any connectivity class F_i, the graph (V, F_i) does not have the same edge-connectivity properties that $G = (V, E)$ did. An edge $e \in F_i$ could have edge-connectivity $\Omega(n)$ in G and 1 in (V, F_i), e.g., a graph with a single (s, t) edge and $\Omega(n)$ parallel paths from s to t of length two. The following definitions by Fung et al. [47] are used to construct a subgraph of G that is slightly larger than F_i that certify connectivity properties of edges in F_i.

Definition 4.2.4 (Π-connected decomposition [47]) *Given an undirected graph G and connectivity parameters λ_e, a decomposition $\mathcal{G} = (G_i = (V, E_i))_{1 \le i \le \Lambda}$ is a sequence of subgraphs of G that satisfy $F_i \subseteq E_i$ for all i with $\Lambda = \max_e \lceil \log \lambda_e \rceil + 1$. Further, for a sequence of parameters $\Pi = (\pi_i)_{1 \le i \le \Lambda}$, the decomposition satisfies Π-connectivity if all edges $e \in F_i$ have edge-connectivity at least π_i in G_i for all i.*

Note that the same edge is allowed to appear in multiple sets E_i.

Definition 4.2.5 (γ-overlap [47]) *For any $\gamma \ge 1$, an undirected graph G and its Π-connected decomposition $\mathcal{G}$ satisfies γ-overlap if for all i and for all cuts $C = \partial_G(S)$,*

$$\sum_{i=0}^{\Lambda} u(C \cap E_i) \cdot \frac{2^{i-1}}{\pi_i} \le \gamma \cdot u(C)$$

4.2.3. Balance and Cut Counting

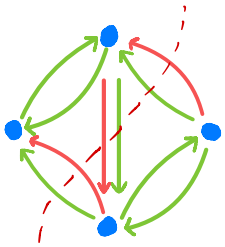
On a complete bipartite graph with edges directed only from left to right, even preserving connectivity requires storing all edges. However it is very imbalanced by the definition below.

Imbalance of Directed Graphs. Preserving various graph properties by subsampling is difficult on general directed graphs. However, sampling algorithms can be successfully applied when the directed graph exhibits some nice structure, for example, they have a similar number of edges crossing a cut in both directions. Eulerian graphs have exactly the same number of edges crossing any cut in both directions, and more pertinently, the residual graph of a flow that is not a $(1 - \varepsilon)$ approximate max flow has at least an $O(\varepsilon)$ fraction of edges crossing any cut in one direction over the other.

Definition 4.2.6 (balance [48]) *Given a directed (multi-)graph $\vec{G} = (V, \vec{E})$, the (im)balance of a cut $\vec{C} = \partial_{\vec{G}}(S)$ is defined as*

$$\beta(\vec{C}) = \frac{u(\vec{C})}{u(\overleftarrow{C})}$$

where $\overleftarrow{C} = \partial_{\vec{G}}(\bar{S})$ is the set of edges in the other direction of the cut.



[48]: Ene et al. (2016), "Routing under balance"

Counting Cut Projections. Karger shows that there are at most $n^{2\alpha}$ distinct cuts whose size is $\leq \alpha\lambda$, where λ is the global mincut. We, like Fung et al. [47], look at larger cuts in the graph and require a tighter bound than the one given by Karger, which requires the following definition.

Definition 4.2.7 (directed k-projection) *Let $\vec{G} = (V, \vec{E})$ be a directed (multi-)graph and let G be the corresponding undirected graph. The k-projection of any cut $\vec{C} = \partial(S)$ in $\vec{G}$ is $\vec{C} \cap H_k$ where*

$$H_k = \left\{\vec{e} \in \vec{E} \,:\, \text{the edge-connectivity of } e \text{ in } G \text{ is at least } k\right\}$$

4.2.4. Data Structures

We also need the following three classic data structures for our results.

Fact 4.2.1 (UnionFind [49]) *There is a data structure that supports the following operations*

- Add(u): *Adds $\{u\}$ to the set of sets in $O(1)$ time.*
- Find(u): *Finds the representative of the set that u belongs to in amortized $O(\alpha(n))$ time.*
- Union(u, v): *Replaces the sets that u and v belong to with their union in $O(1)$ time.*

where n is the number of elements present at that time, and $\alpha(n)$ is the inverse Ackermann function.

UnionFinds are used to maintain incremental trees for NI indices.

[49]: Tarjan et al. (1984), "Worst-case Analysis of Set Union Algorithms"

Fact 4.2.2 (SingleSourceReachability [50]) *There is a data structure that supports the following operations on a graph $\vec{G}$ with n vertices*

- Initialize($\vec{G}, s$): *Initializes a data structure on $\vec{G}$ with special vertex s in $O(m+n)$ time.*
- Insert($e = \vec{uv}$): *Inserts the directed edge $\{e = \vec{uv}\}$ into $\vec{G}$ in amortized $O(1)$ time.*
- Reachable(t): *Returns "True" if t is reachable from s in $\vec{G}$ in amortized $O(1)$ time.*
- GetPath(t): *Returns a directed $s \to t$ path in $\vec{G}$ in time proportional to the length of the path.*

Incremental SingleSourceReachabilitys are used to find augmenting paths in the residual graph.

[50]: Italiano (1986), "Amortized Efficiency of a Path Retrieval Data Structure"

Fact 4.2.3 (BinarySearchTree) *There is a data structure that supports the following operations*

- Initialize: *Initializes the binary search tree data structure.*

BinarySearchTrees are used to quickly find the NI index of each edge.

- INSERT(e, x): *Inserts the key e with value x in amortized $O(\log n)$ time.*
- SEARCH(x): *Returns the largest key e with value $\leq x$ in amortized $O(\log n)$ time.*

where n is the number of items inserted into the data structure at time of operation.

High Probability Bounds. An algorithm satisfies a guarantee *with high probability* if the guarantee holds with probability at least $1 - 4/n$.

In the conference version of this chapter, the results held only against an oblivious adversary. The current version also works against output-adaptive adversaries.

Adversaries. An adversary is said to be *oblivious* if it fixes the update sequence before the algorithm picks its random bits. An adversary is *output-adaptive* if it is allowed to choose the next update/query based on the past updates and the sequence of answers given by the algorithm to past queries. An adversary is *state-adaptive* if it is output-adaptive and also allowed access to the internal state of the algorithm before choosing the next update.

4.3. Technical Overview

[17]: Karger et al. (2015), "Fast Augmenting Paths by Random Sampling from Residual Graphs"

The main idea underlying our result is to dynamize the static algorithm by Karger and Levine [17] (abbrv. KL algorithm) that computes an *exact* max flow in $\widetilde{O}(m + nF^*)$ time. We start by reviewing their static construction and then identify the challenges in the dynamic setting, along with our approach to overcome them. We present the algorithm in full in Section 4.4.

The KL algorithm repeatedly samples $\rho = O(n \log n)$ edges from the residual graph, searches for an augmenting path in this sample, and then augments along this path before resampling again. When the search fails to find an augmenting path, the number of sampled edges is doubled to 2ρ and this process is repeated until the sampled graph becomes as large as the original graph. Denoting the maximum flow value by F^*, they show that at least $F^*/2$ of the augmentations are performed on the sparsest graph with ρ edges, at least $F^*/4$ of the augmentations are performed on the graph with 2ρ edges and so on. After spending $\widetilde{O}(m)$ on pre-processing to compute the sampling probabilities, the total running time thus adds up to $\approx \sum_i (F^*/2^i) \cdot 2^{i-1}\rho$, which gives the required $\widetilde{O}(m + nF^*)$ bound. Crucially, the last augmentation from $F^* - 1$ to F^* is performed on a graph of size $\Omega(m)$.

This final augmentation, which requires a sample of size $\Omega(m)$, effectively invalidates any attempts to use the KL approach for obtaining an incremental *exact* max flow algorithm. To understand why, observe that after every edge insertion, the algorithm needs to check if this edge insertion creates a new $s \to t$ augmenting path in the sampled graph. As we discuss below, this can be done in two ways, but both lead to algorithmic dead ends.

[50]: Italiano (1986), "Amortized Efficiency of a Path Retrieval Data Structure"

We could try to use an incremental single source reachability data structure (e.g., the one by Italiano [50]) to detect augmenting paths in the sampled graph, but since the actual augmentation reverses the direction of edges, we would need to reinitialize the incremental data structure on the sample after each augmentation. Since F^* augmentations are performed on samples of size $\Omega(m)$, this leads to a time bound of $\Omega(mF^*)$. As a concrete example, consider an initially empty bipartite graph $G = (S \cup T, \emptyset)$, with $s \in S$ and $t \in T$. In the first phase, insert all edges between $S \times T \setminus \{t\}$, then in the second phase, insert all the remaining edges of the type $\{(v, t)\}_{v \in S}$. Each edge

insertion in the second phase increases the max flow by 1, and any algorithm based on a KL-type approach with an incremental reachability data structure needs to check an $\Omega(m)$-edge graph to perform each augmentation.

The second approach would be to use a *fully dynamic* directed $s \to t$ reachability data structure. This would solve the above problem, as an augmentation in the residual graph can be simulated using $O(n)$ directed edge deletions and insertions. However, fully dynamic $s \to t$ reachability is known to admit strong conditional lower bounds – for any $\eta > 0$, no algorithm can achieve $O(n^{1-\eta})$ worst-case update time and $O(n^{2-\eta})$ worst-case query time, assuming the OMv conjecture [37].

[37]: Henzinger et al. (2015), "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture"

Interestingly, we show that the first approach can be extended to the incremental setting if we relax our algorithm to maintain a $(1-\varepsilon)$-*approximate* s-t max flow instead of an exact one. The high-level reason why is as follows: To show that at least $F^*/2^i$ of the augmentations are found in the sampled graph of size $2^{i-1}\rho$, Karger and Levine prove that when at least $F^*/2^i$ of the max flow still remains to be augmented, a sampled graph of size $2^{i-1}\rho$ suffices to preserve the existence of an augmenting path whp. For our setting, if we want to maintain a $(1-\varepsilon)$-approximate max flow, the above claim with $i = \log(1/\varepsilon)$ tells us that it suffices to maintain a sparsifier with just $O(n\log(n)/\varepsilon)$ edges incrementally. This then removes the need to augment on an $\Omega(m)$-edge graph which lets us circumvent the above barrier. While incrementally maintaining the sparsifier used by Karger and Levine would be highly non-trivial, we show that we can both efficiently maintain a different sparsifier incrementally, and that the new sparsifier is powerful enough to recover the guarantees required for the KL approach to work (at the cost of a $\log n$ factor in the sparsifier size).

The challenge of incrementally maintaining the KL sparsifier is due to its fragility under edge insertions. The KL sampling depends on a parameter called the *strong connectivity* of an edge [51], and in the incremental setting, a single edge insertion could modify the strong connectivities (and thus the sampling probabilities) of *all* other edges of the graph. As a result, no existing algorithm can even just maintain the sampling probabilities, let alone maintain an actual sample from the corresponding distribution. The same issue arises for other well-known connectivity measures used for sampling such as edge-connectivity [47] and effective resistances [52], and even maintaining approximations to them is quite involved [38, 53–55]. Instead, our key observation is to use *Nagamochi-Ibaraki (NI) indices* [46] for our sampling because of their resilience to incremental updates.

[51]: Benczúr et al. (2015), "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs"

[52]: Spielman et al. (2011), "Graph Sparsification by Effective Resistances"

[38]: Chen et al. (2020), "Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers"

[53]: Durfee et al. (2018), "Fully Dynamic Effective Resistances"

[54]: Durfee et al. (2019), "Fully dynamic spectral vertex sparsifiers and applications"

[55]: Gao et al. (2021), "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao"

[46]: Nagamochi et al. (1992), "A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph"

While the other connectivity parameters mentioned are unique for an edge, the NI index is determined by the forest packing constructed, and different packings gives rise to different indices. This flexibility in choosing the index allows us to maintain the same value even under edge updates, which we crucially use to maintain the sampling probabilities. The resilience property we use is as follows: when inserting an edge, the NI index of all other edges remains the same, and our task is to only determine the NI index of the newly inserted edge efficiently (which then doesn't change in the future). Efficient maintenance of NI indices is discussed in Section 4.5.

While we now maintain a set of sampling probabilities using NI indices, it is unclear how they can be used to replace strong connectivity in the KL algorithm. We extend the general result of Fung et al. [47] on using a wide variety of connectivity parameters for cut sparsification on *undirected* graphs to showing that the same parameters also lead to cut sparsification on balanced directed graphs, which cover residual graphs that arise in our algorithm as a special case. Their results sample each edge independently,

which in our setting leads to a prohibitive $\Omega(m)$ time for sampling. We adapt this to repeated sampling from a probability distribution, where we only spend $O(\log n)$ time per edge for a total $O(n \log^2(n)/\varepsilon)$ edges sampled. We present this result in full detail in Section 4.6.

Finally, we simplify our algorithm by not maintaining an exact sample from the distribution at each time step, but an oversample. Specifically, we obtain a sample of the correct size ($O(n \log^2(n)/\varepsilon)$) right after an augmentation, but between two augmentations, we add edges directly to the sample instead of trying to maintain the distribution. This could blow up the number of edges in the sample to $\Omega(m)$, but since each edge is added this way to at most one sample throughout the algorithm, the total time bound of $\widetilde{O}(m + nF^*/\varepsilon)$ still holds. This also helps us obtain our guarantee against adversaries that know the current flow maintained by the algorithm, since the sample remains independent of the flow: any time the flow is updated, we resample the graph based on NI indices (which are independent of the flow), and between two augmentations, all inserted edges are directly added to the sample.

4.4. Incremental Maximum Flow

We assume that the algorithm is initialized on the empty graph. Our main result in this section is the following:

Theorem 4.1.1 *Given any $\varepsilon \in (0, 1)$, there is an incremental randomized algorithm against output-adaptive adversaries that maintains a $(1-\varepsilon)$-approximate maximum s-t flow f under edge insertions on an undirected uncapacitated n-vertex graph G with high probability in total time $O(m \log(n)\alpha(n) + nF^* \log^3(n)/\varepsilon)$, where m is the number of edge insertions, F^* is the value of the max flow after m insertions, and $\alpha(n)$ is the inverse Ackermann function.*

Each phase ends with the flow value increasing by 1.

We present the algorithm achieving the guarantees in Algorithm 4.1. The algorithm works in phases. At the beginning of a new phase, we sample $O(n \log^2(n)/\varepsilon)$ edges from the residual graph using a specific probability distribution which we will discuss later, and add them to the *sampled graph* H. On top of H, we run an incremental directed single-source reachability algorithm, called D, starting from s. At the beginning, and after each edge insertion, we query D if there exists an $s \to t$ path in the sample. As long as D does not return "Yes", for each edge insertion $e = (u, v)$, we add edges $\vec{uv}$ and $\vec{vu}$ to D. When D returns "Yes", this corresponds to an $s \to t$ path in the residual graph, and we retrieve this path from D, augment along this path in G_f, and start a new phase.

This bounds the number of phases by F^*.

We need to argue about the time bounds and the correctness. For correctness, we show that whenever an edge insertion increases the max flow to a value larger than a $(1-\varepsilon)^{-1}$ factor over the current flow maintained by the algorithm, the existence of an $s \to t$ path in the sampled graph H is guaranteed with high probability. Since H is a subsample of the residual graph G_f, this provides us with an augmenting path in G_f along which we augment the flow. With a union bound over the at most n times when this can happen (since the max flow value is at most n and each augmentation increases the flow by 1), this will imply that at the end of every time step, the algorithm maintains a valid $(1-\varepsilon)$-approximate s-t flow with high probability. To show this claim, we first show that an extension of Fung et al.'s high probability result [47] on independent sampling for cut sparsification on undirected graphs also extends to repeated sampling for cut

Algorithm 4.1: Algorithm for incremental approximate maximum flow on undirected, uncapacitated graphs

Function Initialize(ε):
  $\rho \leftarrow 5390 \cdot n \log^2(n)/\varepsilon$
  $f \leftarrow \emptyset$, $F \leftarrow 0$, $G_f \leftarrow (V, \emptyset)$, $H \leftarrow G_f$
  $D \leftarrow$ SingleSourceReachability.Initialize(H, s)
  $K \leftarrow$ IncNISample.Initialize ▷ Algorithm 4.4
Function Insert($e = (u, v)$):
  Add $\vec{uv}$ and $\vec{vu}$ to H and D
  K.Insert(e)
  if D.Reachable(t) **then**
    $p \leftarrow D$.GetPath(t)
    update f and G_f by augmenting along p, and update $F \leftarrow F + 1$
    reset $H \leftarrow (V, \emptyset)$
    for $i = 1, 2, \ldots, \rho$ **do**
      $(u, v) \leftarrow K$.Sample
      for $\vec{ab} \in \{\vec{uv}, \vec{vu}\}$ **do**
        if $\vec{ab} \in G_f$ **then**
          Add $\vec{ab}$ to H
    reinitialize $D \leftarrow$ SingleSourceReachability.Initialize(H, s)
Function Query(*flow/value*):
  if query = flow **then return** f **else return** F

} add both directions of edge to H.

sparsification on balanced directed graphs. When combined with the fact that the residual graph of a flow that is not a $(1-\varepsilon)$-approximate max flow is $\Omega(\varepsilon)$-balanced, this shows that sampling approximately $n \log^2(n)/\varepsilon$ edges based on the probability distribution discussed below will preserve directed cuts, and thus also the existence of an augmenting path in residual graphs of non-$(1-\varepsilon)$-approximate flows. Our initial sample at the beginning of a phase already satisfies the size requirement above, and since we add further edges directly to the maintained sample, we always oversample the rate required for the augmenting path preservation guarantee.

For the time bounds, assuming that the sampling can be done in time approximately $O(\log n)$ per sample and that the reachability data structure runs in total time proportional to the number of edges in D, we will show that the algorithm runs in $\widetilde{O}(m + nF^*/\varepsilon)$ time. If we were able to maintain the size of the sampled graph at $\widetilde{O}(n/\varepsilon)$ throughout a phase, then the claimed time bound would follow as (1) D runs in time $\widetilde{O}(n/\varepsilon)$ in each phase if the graph has size $\widetilde{O}(n/\varepsilon)$, (2) at the end of each phase the value of the flow increases by 1, and (3) the number of phases is at most F^* as the final max flow is by definition F^*. However, the size of the sampled graph could increase to $\Omega(m)$ if no paths are found by D over all edge insertions, so this argument does not immediately work. Instead the bound follows from a slight modification to the above argument: in each phase, denote an edge to be "old" when it is inserted into H due to sampling, and "new" when it is inserted into H directly on insertion in G. Then the claimed time bound follows since each edge appears as "new" in at most one phase throughout the algorithm, and the previous argument still holds for the "old" edges of each phase since there are at most $\widetilde{O}(n/\varepsilon)$ of them per phase.

Finally, we maintain NI indices incrementally for the probability distribution, which we discuss in Section 4.5. We use the following definition of a phase.

Algorithm 4.2: Algorithm for directed balanced sparsification

Input: Directed graph $\vec{G} = (V, \vec{E})$, connectivity parameters $\{\lambda_e\}_{e \in E}$, parameters $\beta, \gamma \geq 1$, and $\varepsilon < 1$
Output: Sparsified graph $\vec{H} = (V, \vec{F}, w)$

```
ρ ← (128γ(β+1) ln n / 0.38ε²) · Σ_{e∈E} λ_e^{-1}
H ← (V, ∅, w)
for i ∈ 1, 2, ..., ρ do
    Sample an edge e from the probability distribution {p_e = λ_e^{-1} / Σ_e λ_e^{-1}}
    Insert e into H with weight w_e = (ρ · p_e)^{-1} (additively increasing its weight
      if e already exists in H)
```

Definition 4.4.1 (Phase) *Let $t_0 = 0$, and let t_i be the time step when the i^{th} augmenting path is found. Let m_k be the number of edges inserted in time steps $(t_{k-1}, t_k]$. Phase k refers to the computations performed after the $(k-1)^{th}$ augmentation finishes until the k^{th} augmentation ends.*

We also use the following two statements directly, whose proofs can be found in Section 4.5 and Section 4.6 respectively.

This allows quickly resampling H after a phase. Specifically, we spend $O(\log m \alpha(n))$ time per sampled edge.

Lemma 4.4.1 *There is an incremental data structure that maintains an NI forest packing (and thus the NI index ℓ_e of each edge) of an undirected graph under edge insertions in total time $O(m \cdot \log m \cdot \alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function and m is the number of edges in the final graph. At any point of time, it also allows querying for an edge sampled from the probability distribution $\{\ell_e^{-1}/L\}_{e \in E}$ where $L = \sum_e \ell_e^{-1}$ in amortized time $O(\log n)$ for each sample.*

This gives the requirements for cut sparsifying a directed balanced graph.

Theorem 4.4.2 *Let $\vec{G} = (V, \vec{E})$ be any directed (multi-)graph, and G be the underlying undirected graph. Let $\{\lambda_e\}_{e \in E}$ be some connectivity parameters in G, and $\beta, \gamma \geq 1$ and $\varepsilon < 1$ be input parameters. Let $\vec{H} = (V, \vec{F}, w)$ be the random graph with $O(\gamma \beta \ln(n) \cdot \sum_{e \in E} \lambda_e^{-1}/\varepsilon^2)$ edges generated as in Algorithm 4.2.*
If there exists a Π-connected decomposition $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq i \leq \Lambda\}$ of G that satisfies γ-overlap, then for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ of balance at least β^{-1} in $\vec{G}$, it holds simultaneously with probability $\geq 1 - 4/n^2$ that

$$(1 - \varepsilon) \cdot u(\vec{C}) \leq w(\vec{C}) \leq (1 + \varepsilon) \cdot u(\vec{C}).$$

4.4.1. Running time

We first bound the running time within a single phase.

Lemma 4.4.3 *Phase k takes total time $O(m_k \log(n) \alpha(n) + n \log^3(n)/\varepsilon)$, where m_k is the number of edges inserted in Phase k and $\alpha(n)$ is the inverse Ackermann function.*

Proof. Phase k starts with sampling $O(n \log^2(n)/\varepsilon)$ edges from the sampler, where each sample takes time $O(\log n)$ by Lemma 4.4.1. D has a total of $m_k + O(n \log^2(n)/\varepsilon)$ many edges added to it in its entire lifetime, which gives a total update time of $O(m_k + n \log^2(n)/\varepsilon)$. Inserting each edge into the

sampler takes amortized time $O(\log m \cdot \alpha(n))$ by Lemma 4.4.1. Augmenting along a path takes time proportional to its length, which is at most n. □

Lemma 4.4.4 *The algorithm takes total time $O(m \log(n)\alpha(n) + n\log^3(n)F^*/\varepsilon)$ where F^* is the value of the final max flow after all edge insertions.*

The general guarantee follows by summing up the per-phase guarantee over all phases.

Proof. Let K be the total number of phases in the algorithm. Then $K \le F^*$ since each phase increases the value of the flow by 1. Since the set of edges inserted in each phase are disjoint, $\sum_k m_k = m$. By Lemma 4.4.3, phase k takes time $O(m_k \log(n)\alpha(n) + n\log^3(n)/\varepsilon)$. Together, all phases take time

$$\sum_{k=1}^{K} O(m_k \log(n)\alpha(n) + n\log^3(n)/\varepsilon) = O\left(m\log(n)\alpha(n) + n\log^3(n)F^*/\varepsilon\right). \quad \square$$

4.4.2. Correctness

Lemma 4.4.5 *Let f be any s-t flow in an undirected graph G of value F, and let G_f be its corresponding residual graph. Suppose $F \le (1-\varepsilon)F^*$, where F^* is the value of the maximum flow. Then every cut in G_f is at least $\varepsilon/2$-balanced.*

We first show that the residual graph of a non-$(1-\varepsilon)$-approximate max flow is $\varepsilon/2$-balanced. This allows us to use our result on sparsifying balanced directed graphs.

Proof. For any cut $\vec{C} = \partial(S)$ that is not an s-t cut (or a t-s cut), the net flow out of S in G_f is zero, and thus $\vec{C}$ is 1-balanced. All t-s cuts are $\ge$ 1-balanced since the residual graph has more flow in the $s \to t$ direction and thus more capacity in the $t \to s$ direction. Consider an s-t cut $\vec{C}$. Let c be the capacity of the underlying undirected cut C. Since F units of flow cross C in the forward direction $s \to t$, the forward capacity across C in G_f is $c - F$, and the backward capacity is thus $c + F$ since the total capacity is $2c$. Thus, we get

$$\begin{aligned}
\beta(\vec{C}) &= \frac{u(\vec{C})}{u(\overleftarrow{C})} = \frac{c-F}{c+F} \\
&\ge \frac{c-F}{2c} = \frac{1}{2}\cdot\left(1 - \frac{F}{c}\right) && \text{(since } c \ge F\text{)} \\
&\ge \frac{1}{2}\cdot\left(1 - \frac{F}{F^*}\right) && \text{(since } c \ge F^*\text{)} \\
&\ge \frac{\varepsilon}{2} && \text{(since } F \le (1-\varepsilon)F^*\text{)}
\end{aligned}$$

as required. □

Lemma 4.4.6 *Let $\{\ell_e\}_{e\in E}$ be a set of NI indices for an undirected (multi-)graph G with m edges. Then $\sum_e \ell_e^{-1} \le 2n\log m$.*

We then show that the sum of inverse NI indices is not too large.

Proof. Consider the NI forest packing $T_1, T_2, \ldots$ corresponding to the NI indices ℓ_e. Since each forest contains at least one edge, the total number of forests is at most m. As each edge e in T_i has $\ell_e = i$, the sum of NI indices of all edges in forest i contributes at most $\frac{n-1}{i}$ to the total sum. Thus,

$$\sum_{e\in E} \ell_e^{-1} = \sum_{1\le i\le m}\sum_{e\in T_i} \ell_e^{-1} \le \sum_{1\le i\le m} \frac{n-1}{i} \le (n-1)\cdot(2\log m) \le 2n\log m$$

where $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m}$ is the m-th harmonic number, which is upper bounded by $2 \log m$. □

Since the residual graph has two copies of each edge of G, we prove this lemma to show that an NI forest packing of G can be transformed into a packing of the undirected residual graph.

Lemma 4.4.7 *Let $\{\ell_e\}_{e \in E}$ be a set of NI indices for an undirected, simple graph $G = (V, E)$. Let E' be a copy of the edges in E. Then, for the multigraph $H = (V, E \cup E')$ which doubles the edges in E, assigning $h_e = 2\ell_e - 1$ and $h_{e'} = 2\ell_e$ is a set of NI indices for H.*

Proof. If $T_1, T_2, \ldots$ forms an NI forest packing of G, then $T_1, T_1, T_2, T_2, \ldots$ forms an NI forest packing of H. Letting e be the edge in the first copy of T_i and e' be the edge in the second copy gives the lemma. □

We use the result of Fung et al. [47] that shows that an NI forest packing can be used to obtain a decomposition that satisfies Π-connectivity and γ-overlap.

Lemma 4.4.8 ([47]) *Let $T_1, T_2, \ldots$ be an NI forest packing of an undirected graph G leading to NI indices ℓ_e, and let $\mathcal{F} = \{F_i\}$ be the corresponding connectivity classes. Then $E_1 = F_1$ and $E_i = F_{i-1} \cup F_i$ for $i \geq 2$ is a decomposition that satisfies Π-connectivity and γ-overlap for $\pi_i = 2^{i-1}$ and $\gamma = 2$, i.e., for all i and for all cuts $C = \partial_G(S)$,*

$$\sum_{i=0}^{\Lambda} u(C \cap E_i) \leq 2 \cdot u(C).$$

Using these, we will first show that sampling $\Omega(n \log^2(n)/\varepsilon)$ edges suffices to sample at least one edge across all balanced cuts.

Lemma 4.4.9 *Consider an undirected, uncapacitated graph G with some s-t flow f of value F. Let ℓ_e be a NI index of edge e. Let $H' = (V, \vec{F})$ be a sample of $5390 \cdot n \log^2(n)/\varepsilon$ edges chosen i.i.d. with probability $\{\ell_e^{-1} / \sum_e \ell_e^{-1}\}_{e \in G_f}$ from the residual graph G_f. Then it holds with probability $1 - 4/n^2$ that for every directed cut $\vec{C} = \partial(S)$ in G_f of balance at least $\varepsilon/2$, there is an edge $e \in \vec{C}$ that is sampled in H'. In particular, if F^* is the value of the max flow in G and $F \leq (1-\varepsilon)F^*$, then there exists an augmenting path in the sampled graph H' with probability $1 - 4/n^2$.*

Proof. We will first argue that sampling the mentioned number of edges and reweighting them as in Algorithm 4.2 gives a (weighted) cut sparsifier $\vec{H} = (V, \vec{F}, w)$ of G_f. This implies positive weight across every balanced cut in $\vec{H}$. Then, we note that the property of presence of an edge across the cut holds regardless of the presence of weights. Since the sample H' chosen in the lemma is an unweighted version of $\vec{H}$, the lemma follows.

Since there are at most $\binom{n}{2}$ edges in a simple graph G (and thus at most n^2 edges in a residual graph), Lemma 4.4.6 gives that that $\sum_{e \in E} \ell_e^{-1} \leq 4n \log n$. By Lemma 4.4.8, there is a decomposition for NI indices that satisfies Π-connectivity and γ-overlap for $\pi_i = 2^{i-1}$ and $\gamma = 2$. Thus, if one runs Algorithm 4.2 with $\rho = 5390 \cdot n \log^2(n)/\varepsilon$ edges, then by Theorem 4.4.2 one obtains a weighted $(1 + \varepsilon')$-sparsifier $\vec{H} = (V, \vec{F}, w)$ of the residual graph G_f where the parameters are set as

$$\varepsilon' = 1/2, \quad \gamma = 2, \quad \beta = 2/\varepsilon, \quad \sum_e \ell_e^{-1} \leq 4n \log n.$$

Thus, for all cuts $\vec{C}$ in G_f of balance at least $\varepsilon/2$ (Lemma 4.4.5), we have that the weighted sparsifier $\vec{H} = (V, \vec{F}, w)$ from Theorem 4.4.2 satisfies

$$w(\vec{C}) \geq \frac{u(\vec{C})}{2} > 0$$

simultaneously with probability $1 - 4/n^2$. In particular, this also means that at least one edge from S to $\bar{S}$ is sampled in $\vec{H}$ for every directed s-t cut $\partial(S)$.

Defining $H' = (V, \vec{F})$ to be the unweighted graph obtained from $\vec{H}$ by dropping the edge weights, the existence of an edge crossing S to $\bar{S}$ holds in H' as well for all cuts $\partial(S)$ with balance at least $\varepsilon/2$ simultaneously, with probability $1 - 4/n^2$. Since the sampled graph stated in the lemma is generated exactly as in H', the lemma follows. □

Lemma 4.4.10 *At the beginning of any phase k, if $F \leq (1-\varepsilon)F^*_{t_{k-1}}$, then there exists an augmenting path in H with probability $1 - 4/n^2$. Here, F^*_i is the max flow at time i, and t_{k-1} is the time step when the $(k-1)$-st augmentation ends. This guarantee holds against output-adaptive adversaries.*

We first show that right after a sample happens (and before any future edge insertions), the augmenting path guarantee holds w.h.p.

Proof. Recall that at the beginning of a phase, the graph H is sampled according to NI indices in the undirected graph G. We show that the conditions of Lemma 4.4.9 hold, even with an output-adaptive adversary. Note that the sampling *probabilities* are maintained deterministically and are known even to an oblivious adversary. While an output adaptive adversary knows the flow maintained by the algorithm (and thus the current residual graph as well), the sampling according to NI indices is *independent* of the flow maintained by the algorithm and its previous outputs. Lemma 4.4.9 shows the existence of an augmenting path for the residual graph of *any* flow, so no matter which updates were performed in the past and what information the output-adaptive adversary has. Since we assume in the lemma that we are at the beginning of a phase, there have been no edge insertions after the sampling of H. Since we sample an undirected edge and add both directions of that edge into H (lines 15 to 17), this is an oversample of the rate required by Lemma 4.4.9, which only requires adding one direction of the edge. Thus, the claim follows by the guarantees of Lemma 4.4.9. □

Lemma 4.4.11 *Consider an undirected, uncapacitated graph $G = (V, E)$ with some s-t flow f of value F. Let $H' = (V, \vec{F})$ be the graph sampled from G_f as in Lemma 4.4.9, and let $\mathscr{E} = \{X \subseteq (V \times V) \mid X \cap E = \emptyset\}$ be the set of all possible future updates. For any $X \in \mathscr{E}$, define F^*_X to be the max flow in the graph $G_X = (V, E \cup X)$, and define $H'_X = (V, \vec{F} \cup X)$. Then the following guarantee holds with probability at least $1 - 4/n^2$: for every $X \in \mathscr{E}$, if $F \leq (1-\varepsilon)F^*_X$, then there exists an augmenting path in $H'_X = (V, \vec{F} \cup X)$.*

To extend the guarantee to all time steps, we show that Lemma 4.4.9 holds regardless of the future sequence of edge insertions (called X) that happen in the incremental algorithm. An adversary who cannot see the internal randomness of the sample does not get any power by choosing future updates, since the guarantee holds regardless of the choice of future edge insertions.

Proof. By Lemma 4.4.9, we have that with probability at least $1 - 4/n^2$, the initially sampled graph H' of G_f satisfies the following: for any directed s-t cut $\partial(S)$ of balance at least $\varepsilon/2$, there is at least one edge sampled from S to $\bar{S}$. We assume in what follows that H' satisfies this guarantee, thereby conditioning on an event of probability at least $1 - 4/n^2$. Fix an arbitrary $X \in \mathscr{E}$ such that $F \leq (1-\varepsilon)F^*_X$. We will show that the augmenting path guarantee holds in H'_X, which completes the proof. Note that the augmenting path guarantee is equivalent to showing that for all directed s-t cuts $\partial(S)$ in H'_X, there is at least one edge from S to $\bar{S}$. Let $\vec{C} = \partial(S)$ be an arbitrary

directed s-t cut. By Lemma 4.4.5, $\vec{C}$ has balance at least $\varepsilon/2$ in $(G \cup X)_f$. If $\vec{C}$ had balance at least $\varepsilon/2$ even in G_f, then there is at least one edge sampled across this cut in H' by the conditioning above. If not, then the balance of $\vec{C}$ changed between G_f and $(G \cup X)_f$, which means that an edge was inserted across $\vec{C}$ in X and, thus, the undirected edge is also added to H'_X. In either case, $\vec{C}$ has an edge crossing the cut in H'_X. Since the choice of cut was arbitrary, this proves the claim. □

Using this, we argue that the algorithm is correct w.h.p. within a phase, regardless of the updates performed within that phase.

Lemma 4.4.12 *Fix a phase k. Let i be the first time step within the phase when $F < (1-\varepsilon)F_i^*$, where F_i^* is the value of the max flow at time i. Then there exists an augmenting path in H with probability $1 - 4/n^2$. The guarantee holds even if an output-adaptive adversary inserts edges in this phase.*

Proof. For phase k, we use Lemma 4.4.11 to argue that the guarantee holds in this phase, for any sequence of future edge insertions. In particular, let i be the first time step within the phase when $F < (1-\varepsilon)F_i^*$. Let X be the sequence of edge insertions in this phase, the augmenting path guarantee holds in the graph H'_X with probability $1 - 4/n^2$ regardless of the identity of X. Since the graph H'_X from Lemma 4.4.11 is the same as the sampled graph H maintained by Algorithm 4.1 after inserting the edges in X in this phase, the lemma holds. □

Finally, we take a simple union bound over all phases to extend the guarantee to all time steps.

Lemma 4.4.13 *Algorithm 4.1 maintains a $(1-\varepsilon)$-approximate s-t max flow at all times with probability $1 - 4/n$ against an output-adaptive adversary.*

Proof. In any phase k, Lemma 4.4.12 ensures that an augmenting path exists in the sample at the first time step i when $F < (1-\varepsilon)F_i^*$ with probability $1 - 4/n^2$. Each time an augmenting path is found, F increases by 1. Thus, we need to invoke Lemma 4.4.12 for at most $F^* \leq n$ phases. Taking a union bound over all the phases, we get that the approximation guarantee holds throughout the entire insertion sequence with probability $1 - 4/n$. □

4.5. Incremental Nagamochi-Ibaraki indices

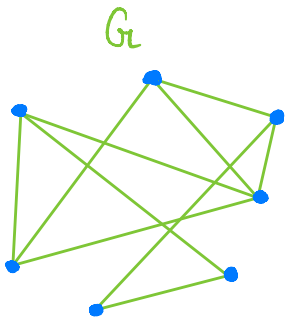
In this section, we show how to incrementally maintain (and sample from) Nagamochi-Ibaraki indices ℓ_e for each edge e. The main result of this section is the following:

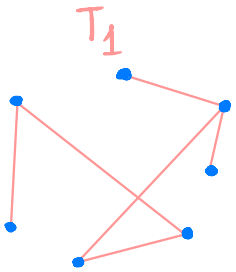
Lemma 4.4.1 *There is an incremental data structure that maintains an NI forest packing (and thus the NI index ℓ_e of each edge) of an undirected graph under edge insertions in total time $O(m \cdot \log m \cdot \alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function and m is the number of edges in the final graph. At any point of time, it also allows querying for an edge sampled from the probability distribution $\{\ell_e^{-1}/L\}_{e \in E}$ where $L = \sum_e \ell_e^{-1}$ in amortized time $O(\log n)$ for each sample.*

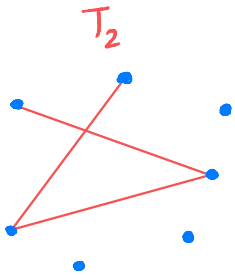
We maintain Nagamochi-Ibaraki indices instead of other sampling parameters because of their stability under incremental updates. When an edge $e = (u, v)$ is inserted into a graph G, it could change other connectivity parameters (such as edge-connectivity, strong-connectivity, or the effective resistance) of *every existing edge* in the graph. However, the NI index of every other edge remains the same on edge insertion, which is a very desirable

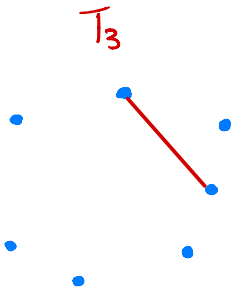
property for dynamic algorithms. This happens because while other graph properties are unique for an edge in a given graph, the NI index depends heavily on the particular forest packing used. It could vary wildly for the same edge, and could range from 1 up to n depending on the packing.

The incremental algorithm involves maintaining a collection of forests, where each forest is a union-find data structure on the vertices currently in that forest. On an edge insertion, we binary search across the forests to find the first one where its endpoints are disconnected, and insert the edge into that forest (and also adding the endpoints to the forest or initializing a new forest if necessary).









Algorithm 4.3: Maintaining Incremental Nagamochi-Ibaraki Indices (IncNIIndex)

```
1  Function Initialize:
2      k ← 0                                          ▷ Number of forests maintained
3      tree ← []                                      ▷ Collection of forests
4      last_tree[v] ← 0 for all v ∈ V                 ▷ last forest that v belongs to
5  Function Insert(e = (u, v)):
6      i ←FindTree(u, v, min(last_tree(u), last_tree(v)))) ▷ find correct forest
         for e
7      if i > k then k += 1, tree[i] ← UnionFind.Initialize ▷ add new forest
8      for x ∈ {u, v} do
9          if i > last_tree(x) then last_tree[x] += 1, tree[i].Add(x)  ▷ add x to
             forest i
10     tree[i].Union(u, v)                            ▷ connect u and v in forest i
11     return i
12 Function FindTree(u, v, upper_bound):
13     L ← 0, R ← upper_bound
14     while L ≤ R do
15         M ← ⌊(L + R)/2⌋                            ▷ binary search for forest
16         if IsConnected(u, v, M) then L ← M + 1 else R ← M − 1
17     return L
18 Function IsConnected(u, v, i):
19     if i = 0 then return True else return tree[i].Find(u) = tree[i].Find(v)
```

Due to the subtleties of implementing this, we have presented the data structure in full detail. In particular, if we naively "initialized" a new forest by inserting *all* the vertices into the forest, then this would lead to a running time of $\Omega(n)$ per forest, which could be prohibitive. For the extreme case where n parallel edges are inserted between the same two vertices u and v, this necessitates initializing $\Omega(n)$ forests, which leads to $\Omega(n^2)$ total time for the naive implementation, as opposed to adding a vertex to a forest only when necessary (as done below with the *last_tree* variable).

Lemma 4.5.1 *Algorithm 4.3 maintains the NI forest packing at all times, and runs in total time* $O(m \cdot \log m \cdot \alpha(n))$.

Proof. We first argue about correctness by induction. It is clearly true before any edge insertions. Suppose it was true until k insertions. Let $e = (u, v)$ be the $(k+1)$-th inserted edge. If i is the index of the tree returned by FindTree, then it satisfies the property that for all $j < i$, vertices u and v are connected in forest $tree[j]$, and that they are not connected in forest $tree[i]$. Thus connecting u and v in forest $tree[i]$ and setting the index of the edge to i maintains correctness after the $(k+1)$-th edge as well.

Next, we argue about the time taken by the algorithm. If there are m edges in the graph currently, then the algorithm maintains $\leq m$ forests. Thus the binary search takes $O(\log m)$ iterations. In each iteration, it queries a union-find data structure if two vertices are connected, which takes $O(\alpha(n))$ amortized time. Union of two elements in, insertion of new elements into, and initialization of a union-find data structure takes constant time. Thus the algorithm runs in total time $O(m \cdot \log m \cdot \alpha(n))$. □

We quickly recall an example of a binary search tree insertion sequence before we prove the next lemma. In a binary search tree, if there are three consecutive insertions of (key, value) pairs $(a, 0), (b, 1/3)$, and $(c, 1/2)$, then searching for any $z \in [0, 1/3)$ returns a, searching for $z \in [1/3, 1/2)$ returns b, and searching for $z \in [1/2, \infty)$ returns c.

Algorithm 4.4: Incremental Nagamochi-Ibaraki sampling (IncNISample)

```
Function Initialize:
    X ← IncNIIndex.Initialize
    Y ← BinarySearchTree.Initialize
    L ← 0
Function Insert(e = (u, v)):
    ℓ_e ← X.Insert(e)
    Y.Insert(e, L)
    L ← L + ℓ_e^{-1}
Function Sample:
    z ← uniform random number in [0, L]
    e ← Y.Search(z)
    return e
```

Lemma 4.5.2 *Algorithm 4.4 maintains a sample from the correct distribution at all times, each insertion takes time $O(\log(m)\alpha(n))$ and each sample takes time $O(\log n)$.*

Proof. Let $e_1, \ldots, e_m$ be the sequence of edge insertions performed, and let ℓ_{e_i} be their corresponding NI indices obtain from Algorithm 4.3. The ℓ_e values obtained from Algorithm 4.3 are correct by the guarantees of Lemma 4.5.1. Define $s_1 = 0$ and $s_i = \sum_{1 \leq j < i} \ell_{e_j}^{-1}$ be the sum of inverse NI indices of all edges inserted until e_{i-1}. We will show that at any time k, for all $i \in \{1, \ldots, k\}$, searching for any $z \in [s_i, s_{i+1}] \subseteq [0, s_{k+1}]$ returns key e_i in the binary search tree. This then gives correctness of the algorithm since, after the k-th insertion, the edge e_i is sampled with probability $(s_{i+1} - s_i)/L = \ell_{e_i}^{-1}/s_{k+1}$ since $L = s_{k+1}$ after k edge insertions.

The statement is clearly true after the first edge insertion. Suppose it was true until $k-1$ insertions. Since the key e_{k-1} had value s_{k-1}, searching for any $z \in [s_{k-1}, \infty)$ would have given key e_{k-1}. Let e_k be the k-th inserted edge, which is inserted with value s_k. Thus, for all $z \in [s_k, \infty)$, the search returns key e_k, and for $z \in [s_{k-1}, s_k)$, the search returns key e_{k-1}. As $s_k - s_{k-1} = \ell_{e_{k-1}}^{-1}$, $s_{k+1} - s_k = \ell_{e_k}^{-1}$, and the other edges e_j for $j < k-1$ are not affected by this insertion, the claim follows.

For time bounds, note that each edge insertion into Algorithm 4.3 takes time $O(\log(m)\alpha(n))$, each insertion into the binary search tree with length ℓ_e^{-1} takes time $O(\log n)$, and each sample takes time $O(\log n)$ as well, given that the random number is generated in time $O(\log n)$. □

Lemma 4.4.1 now follows from Lemma 4.5.1 and Lemma 4.5.2.

4.6. Balanced sparsification

[56]: Cen et al. (2021), "Sparsification of Directed Graphs via Cut Balance"

We remark that while Cen et al. [56] obtain a cut sparsification result for balanced directed graphs, their result only works with strong connectivity and cannot be used with NI indices, which we need for our algorithm. In this section, we show the following result.

Theorem 4.4.2 *Let $\vec{G} = (V, \vec{E})$ be any directed (multi-)graph, and G be the underlying undirected graph. Let $\{\lambda_e\}_{e \in E}$ be some connectivity parameters in G, and $\beta, \gamma \geq 1$ and $\varepsilon < 1$ be input parameters. Let $\vec{H} = (V, \vec{F}, w)$ be the random graph with $O\big(\gamma\beta \ln(n) \cdot \sum_{e \in E} \lambda_e^{-1}/\varepsilon^2\big)$ edges generated as in*

Algorithm 4.2.
If there exists a Π-connected decomposition $\mathcal{G} = \{G_i = (V, E_i) : 1 \le i \le \Lambda\}$ of G that satisfies γ-overlap, then for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ of balance at least β^{-1} in $\vec{G}$, it holds simultaneously with probability $\ge 1 - 4/n^2$ that

$$(1-\varepsilon)\cdot u(\vec{C}) \le w(\vec{C}) \le (1+\varepsilon)\cdot u(\vec{C}).$$

This is an extension of the result of Fung et al. [47] to balanced directed graphs, with the sampling scheme changed from sampling each edge independently with probability $\approx \text{polylog}(n)/(\lambda_e \varepsilon^2)$ to sampling $\approx n\,\text{polylog}(n)/\varepsilon^2$ edges with probability distribution proportional to λ_e^{-1}. The proof proceeds by reducing the problem to showing concentration for each connectivity class where the connectivity parameters are close to each other. Inside each connectivity class F_i, the cuts are then collected by the size of the underlying undirected cut in G_i and concentration is shown for each collection separately. This involves combining the concentration for each cut in the collection, along with a cut-counting argument similar to the one by Fung et al. [47] (which is an extension of Karger's cut-counting). Finally, the concentration of each cut is shown using a standard Chernoff bound.

Concretely, we will show later that the following concentration holds for every connectivity class, but first use it to prove Theorem 4.4.2.

Lemma 4.6.1 *It holds for all $i \in [\Lambda]$ and for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ simultaneously with probability at least $1 - 4/n^2$ that*

$$\left|w(\vec{C}\cap F_i) - u(\vec{C}\cap F_i)\right| \le \frac{\varepsilon}{2}\cdot\left(\frac{u(C\cap E_i)\cdot 2^{i-1}}{\pi_i\cdot\gamma\cdot(\beta+1)} + u(\vec{C}\cap F_i)\right)$$

where $C = \partial_G(S)$ is the underlying undirected cut of $\vec{C}$ in G.

Since it is hard to prove concentration when the connectivity varies wildly across edges, we partition them into classes with connectivities close to each other first.

Proof of Theorem 4.4.2. We group the edges by the probability it is picked, and show concentration for each such connectivity class separately. With probability $\ge 1 - 4/n^2$, we have for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ of balance β^{-1},

$$\begin{aligned}
\left|w(\vec{C}) - u(\vec{C})\right| &= \left|\sum_{i=0}^{\Lambda} w(\vec{C}\cap F_i) - \sum_{i=0}^{\Lambda} u(\vec{C}\cap F_i)\right| \\
&\le \sum_{i=0}^{\Lambda}\left|w(\vec{C}\cap F_i) - u(\vec{C}\cap F_i)\right| && \text{(since } |a+b| \le |a| + |b|\text{)} \\
&\le \frac{\varepsilon}{2}\cdot\left(\sum_{i=0}^{\Lambda}\frac{u(C\cap E_i)\cdot 2^{i-1}}{\pi_i\cdot\gamma\cdot(\beta+1)} + \sum_{i=0}^{\Lambda} u(\vec{C}\cap F_i)\right) && \text{(by Lemma 4.6.1)} \\
&\le \frac{\varepsilon}{2}\cdot\left(\frac{u(C)}{\beta+1} + \sum_{i=0}^{\Lambda} u(\vec{C}\cap F_i)\right) && \text{(by } \gamma\text{-overlap)} \\
&\le \frac{\varepsilon}{2}\cdot\left(\frac{u(C)}{\beta+1} + u(\vec{C})\right) && \text{(since } \mathcal{F} = \{F_1\} \text{ is a partition of } E\text{)} \\
&= \frac{\varepsilon}{2}\cdot\left(\frac{u(\vec{C}) + u(\overleftarrow{C})}{\beta+1} + u(\vec{C})\right) \\
&\le \varepsilon\cdot u(\vec{C}) && \text{(since } u(\overleftarrow{C}) \le \beta\cdot u(\vec{C})\text{)}
\end{aligned}$$

as required. □

To show Lemma 4.6.1, we partition the cuts in a connectivity class based on

the size of the underlying undirected cut, and show concentration in each of them separately. Formally, we will show later that

We again collect cuts into groups with undirected cut close to each other.

Lemma 4.6.2 *Let $\mathscr{C}_{ij}$ be the collection of all cuts $\vec{C} = \partial_{\vec{G}}(S)$ such that for the corresponding undirected cut $C = \partial_G(S)$ in G,*

$$\pi_i \cdot 2^j \leq u(C \cap E_i) \leq \pi_i \cdot 2^{j+1} - 1$$

Then for all cuts $\vec{C}$ in $\mathscr{C}_{ij}$ simultaneously, it holds with probability $1 - 2/n^{4\cdot 2^j}$ that

$$\left|w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i)\right| \leq \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} + u(\vec{C} \cap F_i)\right)$$

We first use this to prove Lemma 4.6.1.

Lemma 4.6.1 *It holds for all $i \in [\Lambda]$ and for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ simultaneously with probability at least $1 - 4/n^2$ that*

$$\left|w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i)\right| \leq \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} + u(\vec{C} \cap F_i)\right)$$

where $C = \partial_G(S)$ is the underlying undirected cut of $\vec{C}$ in G.

Proof. We will show that the statement holds for any fixed i with probability at least $1 - 4/n^4$. Since $\mathscr{F} = \{F_i\}$ is a partition of the edges, and since there are at most n^2 edges in the graph, a union bound over all i with non-empty F_i gives the lemma.

Fix any $i \in [\Lambda]$. For all cuts $\vec{C}$ with no edge in class F_i, the claim holds with probability 1. In what follows, we only consider cuts that have at least one edge in class F_i.

We concentrate on the graph G_i, and only consider cuts that contain at least π_i edges since every edge in F_i has connectivity at least π_i in G_i by Π-connectivity. Let $\mathscr{C}_{ij}$ be the collection of all cuts $\vec{C} = \partial_{\vec{G}}(S)$ such that for the corresponding undirected cut $C = \partial_G(S)$ in G,

$$\pi_i \cdot 2^j \leq u(C \cap E_i) \leq \pi_i \cdot 2^{j+1} - 1$$

Then for all $\vec{C} \in \mathscr{C}_{ij}$, it holds with probability at least $1 - 2/n^{4\cdot 2^j}$ that

(by Lemma 4.6.2) $$\left|w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i)\right| \leq \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} + u(\vec{C} \cap F_i)\right)$$

Thus the probability that there exists a j for which there exists a cut $\vec{C} \in \mathscr{C}_{ij}$ where the bound does not hold is at most

$$\frac{2}{n^4} \cdot \left(\frac{1}{n^{2^0}} + \frac{1}{n^{2^1}} + \ldots\right) \leq \frac{4}{n^4}$$

as required. □

To show Lemma 4.6.2, we show the bound for a single cut in Lemma 4.6.3 and use this with a directed cut counting argument as in Lemma 4.6.4.

Lemma 4.6.3 *For any single cut $\vec{C}$ in $\mathscr{C}_{ij}$, it holds with probability $1-2/n^{8\cdot 2^j}$ that*

$$\left|w(\vec{C}\cap F_i)-u(\vec{C}\cap F_i)\right|\le\frac{\varepsilon}{2}\cdot\left(\frac{u(C\cap E_i)\cdot 2^{i-1}}{\pi_i\cdot\gamma\cdot(\beta+1)}+u(\vec{C}\cap F_i)\right)$$

For a single cut, this bound will follow from a Chernoff bound.

Lemma 4.6.4 *Let $\vec{G}=(V,\vec{E})$ be a directed (multi-)graph, and G be the underlying undirected graph. Let $k\ge\lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $\vec{C}=\partial_{\vec{G}}(S)$ in $\vec{G}$ such that $u(C)\le\alpha k$ for the corresponding undirected cut $C=\partial_G(S)$, and let $\mathscr{C}_\alpha^{\downarrow k}$ be the set of all directed k-projections of these cuts. Then $|\mathscr{C}_\alpha^{\downarrow k}|\le 2\cdot n^{2\alpha}$.*

In the proof of Lemma 1.6.2, there will be many directed cuts $\vec{C}$ that we will need to argue about which all have the same $\vec{C}\cap F_i$. This lemma bounds the number of different $\vec{C}\cap F_i$s that arise.

We first use them to show Lemma 4.6.2.

Lemma 4.6.2 *Let $\mathscr{C}_{ij}$ be the collection of all cuts $\vec{C}=\partial_{\vec{G}}(S)$ such that for the corresponding undirected cut $C=\partial_G(S)$ in G,*

$$\pi_i\cdot 2^j\le u(C\cap E_i)\le\pi_i\cdot 2^{j+1}-1$$

Then for all cuts $\vec{C}$ in $\mathscr{C}_{ij}$ simultaneously, it holds with probability $1-2/n^{4\cdot 2^j}$ that

$$\left|w(\vec{C}\cap F_i)-u(\vec{C}\cap F_i)\right|\le\frac{\varepsilon}{2}\cdot\left(\frac{u(C\cap E_i)\cdot 2^{i-1}}{\pi_i\cdot\gamma\cdot(\beta+1)}+u(\vec{C}\cap F_i)\right)$$

Proof. For any single cut $\vec{C}$ in $\mathscr{C}_{ij}$, with probability at least $1-2/n^{8\cdot 2^j}$,

$$\left|w(\vec{C}\cap F_i)-u(\vec{C}\cap F_i)\right|\le\frac{\varepsilon}{2}\cdot\left(\frac{u(C\cap E_i)\cdot 2^{i-1}}{\pi_i\cdot\gamma\cdot(\beta+1)}+u(\vec{C}\cap F_i)\right)\qquad\text{(by Lemma 4.6.3)}$$

At this point, we would like to take a union bound over all the cuts in $\mathscr{C}_{ij}$. Ideally, we would like the number of cuts in this set to grow as $n^{c\cdot 2^j}$. However, the number of directed cuts $\vec{C}$ that arise might be much larger than that. However, note that the inequality only depends on $\vec{C}\cap F_i$, and not $\vec{C}$. We thus focus on working with the distinct subsets $\vec{C}\cap F_i$ that arise.

We first show that the number of *distinct directed cuts $\vec{C}\cap F_i$ that are encountered* is still $O(n^{c\cdot 2^j})$ using Lemma 4.6.4. In particular, we apply Lemma 4.6.4 on the graph $G_i=(V,E_i)$, setting $k=\pi_i$ and $\alpha k=\pi_i\cdot 2^{j+1}-1$. The total number of distinct subsets $\vec{C}\cap H_k$ that arise from cuts in $\mathscr{C}_{ij}$ is at most

$$2n^{2\alpha}<2n^{4\cdot 2^j}$$

where H_k is the subgraph of G_i of all edges with edge-connectivity at least π_i. Since each edge in F_i has edge-connectivity at least π_i by Π-connectivity, we have that $F_i\subseteq H_k$. Every non-empty subset of the form $\vec{C}\cap F_i$ has a corresponding subset of the form $\vec{C}\cap H_k$ that is counted above, and for every $\vec{C}\cap H_k$ counted above, there is at most one non-empty subset of the form $\vec{C}\cap F_i$, which proves the claim.

To simplify notation, we use the following definition for the next claim. Call a subset of directed edges $\vec{R}\subseteq\vec{E}$ to be *bad* for a directed cut $\vec{C}\subseteq\vec{E}$ if

$\vec{C} \cap F_i = \vec{R}$ and

$$\left|w(\vec{R}) - u(\vec{R})\right| > \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{R}) \right)$$

Note that if $\vec{R}$ is bad for any particular directed cut $\vec{C}$, then it is also bad for

$$\vec{D} = \operatorname*{argmin}_{\vec{C}' : (\vec{C}' \cap F_i = \vec{R}) \wedge (\vec{C}' \in \mathscr{C}_{ij})} u(C' \cap E_i)$$

since

$$\left|w(\vec{R}) - u(\vec{R})\right| > \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{R}) \right) \geq \frac{\varepsilon}{2} \cdot \left(\frac{u(D \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{R}) \right)$$

where the first inequality is because $\vec{R}$ is bad for $\vec{C}$, and the second inequality is by choice of $\vec{D}$.

Thus, the probability we need to bound is

$$\begin{aligned}
& \Pr[\exists \text{ a cut } \vec{C} \in \mathscr{C}_{ij} \text{ such that } \vec{C} \cap F_i \text{ is bad for } \vec{C}] \\
(\text{since } \vec{C} \cap F_i \text{ is such an } \vec{R}) \quad & = \Pr[\exists \text{ an } \vec{R} \subseteq \vec{E} \text{ and a cut } \vec{C} \in \mathscr{C}_{ij} \text{ such that } \vec{R} \text{ is bad for } \vec{C}] \\
(\text{by a union bound}) \quad & \leq \sum_{\vec{R} \subseteq \vec{E}} \Pr[\exists \text{ a cut } \vec{C} \in \mathscr{C}_{ij} \text{ such that } \vec{R} \text{ is bad for } \vec{C}] \\
(\text{since } \vec{R} \text{ bad for } \vec{C} \implies \vec{R} \text{ bad for } \vec{D}) \quad & = \sum_{\vec{R} \subseteq \vec{E}} \Pr\left[\vec{R} \text{ is bad for } \vec{D} = \operatorname*{argmin}_{\vec{C}' : (\vec{C}' \cap F_i = \vec{R}) \wedge (\vec{C}' \in \mathscr{C}_{ij})} u(C \cap E_i) \right] \\
(\text{by Lemma 4.6.3}) \quad & \leq \sum_{\vec{R} \subseteq \vec{E}} \frac{2}{n^{8 \cdot 2^j}} \\
& \leq 2n^{4 \cdot 2^j} \cdot \frac{2}{n^{8 \cdot 2^j}} \leq \frac{4}{n^{4 \cdot 2^j}}
\end{aligned}$$

where the penultimate inequality holds because the number of distinct $\vec{R}$s that can be of the form $\vec{C} \cap F_i$ is at most $2n^{4 \cdot 2^j}$. □

We use a Chernoff bound of the following form to prove Lemma 4.6.3.

We get this by a simple extension of the Chernoff bound as given by Fung et al. [47].

Lemma 4.6.5 *Let $X_1, \ldots, X_n$ be n independent random variables that take values in the range $[0, M]$ for some $M > 0$. Then, for any $\varepsilon \in (0, 1)$ and $N \geq E[\sum_i X_i]$, we have*

$$\Pr\left[\left| \sum_{i=1}^{n} X_i - E\left[\sum_{i=1}^{n} X_i \right] \right| > \varepsilon N \right] \leq 2 \exp(-0.38 \varepsilon^2 N / M)$$

Lemma 4.6.3 *For any single cut $\vec{C}$ in $\mathscr{C}_{ij}$, it holds with probability $1 - 2/n^{8 \cdot 2^j}$ that*

$$\left|w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i)\right| \leq \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

Proof. For each round $j \in \{1, 2, \ldots \rho\}$, let X_j be the following random variable

$$X_j = \begin{cases} w_e = (\rho \cdot p_e)^{-1} & \text{if an edge } e \in \vec{C} \cap F_i \text{ is chosen in round } j \\ 0 & \text{if an edge } e \notin \vec{C} \cap F_i \text{ is chosen in round } j \end{cases}$$

Defining $X = \sum_{j=1}^{\rho} X_j$, we have that $X = w(\vec{C} \cap F_i)$ and $E[X] = u(\vec{C} \cap F_i)$. Since $\lambda_e < 2^{i+1}$ for any edge $e \in F_i$, we have that

$$p_e \geq \frac{\lambda_e^{-1}}{\sum_{e \in E} \lambda_e^{-1}} \geq \frac{1}{2^{i+1} \cdot \sum_{e \in E} \lambda_e^{-1}}$$

This gives an upper bound on the weight as

$$\max_j |X_j| = \max_{e \in \vec{C} \cap F_i} w_e = \max_{e \in \vec{C} \cap F_i} \frac{1}{\rho \cdot p_e} \leq \frac{2^{i+1} \cdot \sum_{e \in E} \lambda_e^{-1}}{\rho}$$

We define this quantity to be M, and apply Lemma 4.6.5 to the random variables X_i using the following explicit values of M and N.

$$M = \left(\frac{128\gamma(\beta+1)\ln n}{0.38 \cdot 2^{i+1}\varepsilon^2} \right)^{-1} \qquad \text{and} \qquad N = u(\vec{C} \cap F_i) + \frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)}$$

It trivially holds that

$$N = u(\vec{C} \cap F_i) + \frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} \geq u(\vec{C} \cap F_i) = E[X]$$

Thus

$$\begin{aligned}
&\Pr\left[\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| > \frac{\varepsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} + u(\vec{C} \cap F_i) \right) \right] \\
&\leq 2 \exp\left(-0.38 \cdot \left(\frac{\varepsilon}{2} \right)^2 \cdot \left(\frac{128\gamma(\beta+1)\ln n}{0.38 \cdot 2^{i+1}\varepsilon^2} \right) \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} + u(\vec{C} \cap F_i) \right) \right) && \text{(by Lemma 4.6.5)} \\
&\leq 2 \exp\left(-0.38 \cdot \left(\frac{\varepsilon}{2} \right)^2 \cdot \left(\frac{128\gamma(\beta+1)\ln n}{0.38 \cdot 2^{i+1}\varepsilon^2} \right) \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta+1)} \right) \right) && \text{(by dropping } u(\vec{C} \cap F_i)\text{)} \\
&\leq 2 \exp\left(-8 \cdot \frac{u(C \cap E_i) \cdot \ln n}{\pi_i} \right) \\
&\leq 2 \exp\left(-8 \cdot 2^j \cdot \ln n \right) && \text{(since } \vec{C} \in \mathscr{C}_{ij} \implies u(C \cap E_i) \geq \pi_i \cdot 2^j\text{)}
\end{aligned}$$

as required. □

We use the following version of Chernoff bound by Fung et al. [47] to obtain Lemma 4.6.5.

Lemma 4.6.6 ([47]) *Let $Y_1, \ldots, Y_n$ be n independent random variables each taking values in $[0,1]$. Let $\mu = E[\sum_i Y_i]$. Then for all $\delta > 0$, we have*

$$\Pr\left[\left| \sum_{i=1}^{n} Y_i - \mu \right| > \delta\mu \right] \leq 2 \exp\left(-0.38 \min\{1, \delta\} \delta\mu \right)$$

Lemma 4.6.5 *Let $X_1, \ldots, X_n$ be n independent random variables that take values in the range $[0, M]$ for some $M > 0$. Then, for any $\varepsilon \in (0,1)$ and $N \geq E[\sum_i X_i]$, we have*

$$\Pr\left[\left| \sum_{i=1}^{n} X_i - E\left[\sum_{i=1}^{n} X_i \right] \right| > \varepsilon N \right] \leq 2 \exp(-0.38\varepsilon^2 N/M)$$

Proof. Define $Y_i = M^{-1} \cdot X_i$. Then $Y_i \in [0,1]$ and $\mu = E[\sum_i Y_i] = M^{-1} \cdot$

$E[\sum_i X_i]$. Applying Lemma 4.6.6 for $\delta = \frac{\varepsilon N}{E[\sum_i X_i]}$, we get that

$$\Pr\left[\left\|\sum_{i=1}^{n} X_i - E\left[\sum_{i=1}^{n} X_i\right]\right\| > \varepsilon N\right] \leq 2\exp\left(-0.38\min\{1,\delta\}\delta\mu\right)$$

If $\delta \leq 1$, then using the fact that $N \geq E[\sum_i X_i]$, the bound becomes

$$2\exp\left(-0.38\cdot\left(\frac{\varepsilon N}{E[\sum_i X_i]}\right)^2 \cdot M^{-1}\cdot E\left[\sum_i X_i\right]\right) \leq 2\exp\left(-0.38\varepsilon^2 N/M\right)$$

and if $\delta > 1$, then the bound becomes

$$2\exp\left(-0.38\cdot\left(\frac{\varepsilon N}{E[\sum_i X_i]}\right)\cdot M^{-1}\cdot E\left[\sum_i X_i\right]\right) \leq 2\exp\left(-0.38\varepsilon N/M\right)$$

where the final bound follows since $\varepsilon^2 < \varepsilon$ for $\varepsilon \in [0,1]$. □

Finally, we use the undirected projection result of Fung et al. [47] to prove Lemma 4.6.4.

This is proved by Fung et al. [47] using a splitting off procedure by Mader [57].

[57]: Mader (1978), "A Reduction Method for Edge-Connectivity in Graphs"

Lemma 4.6.7 ([47]) *Let $G = (V, E)$ be an undirected (multi-)graph. Let $k \geq \lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $C = \partial_G(S)$ in G such that $u(C) \leq \alpha k$, and let $\mathscr{K}_\alpha^{\downarrow k}$ be the set of all undirected k-projections of these cuts, where the undirected k-projection of a cut $C = \partial_G(S)$ is $C \cap H_k$ and H_k is the set of edges with edge-connectivity at least k. Then $|\mathscr{K}_\alpha^{\downarrow k}| \leq n^{2\alpha}$.*

Lemma 4.6.4 *Let $\vec{G} = (V, \vec{E})$ be a directed (multi-)graph, and G be the underlying undirected graph. Let $k \geq \lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $\vec{C} = \partial_{\vec{G}}(S)$ in $\vec{G}$ such that $u(C) \leq \alpha k$ for the corresponding undirected cut $C = \partial_G(S)$, and let $\vec{\mathscr{C}}_\alpha^{\downarrow k}$ be the set of all directed k-projections of these cuts. Then $|\vec{\mathscr{C}}_\alpha^{\downarrow k}| \leq 2\cdot n^{2\alpha}$.*

Proof. We first apply Lemma 4.6.7 on the undirected graph G. Consider the set of all undirected cuts $C = \delta(S, \bar{S})$ in G such that $u(C) \leq \alpha k$. Lemma 4.6.7 gives that there are at most $n^{2\alpha}$ undirected k-projections of such cuts, i.e., $|\mathscr{K}_\alpha^{\downarrow k}| \leq n^{2\alpha}$. We will show that $|\vec{\mathscr{C}}_\alpha^{\downarrow k}| \leq 2\cdot|\mathscr{K}_\alpha^{\downarrow k}|$, which gives the lemma.

We define a function from the first set to the second set where each image has a pre-image set of size at most two. Recall that H_k is the set of edges in G of edge-connectivity at least k. We map each directed projection $\vec{C} \cap H_k$ in $\vec{\mathscr{C}}_\alpha^{\downarrow k}$ to the corresponding undirected projection $C \cap H_k$. The latter belongs to $\mathscr{K}_\alpha^{\downarrow k}$ since $u(C) \leq \alpha k$ by definition. For each $C \cap H_k \in \mathscr{K}_\alpha^{\downarrow k}$, there are at most two directed projections that map to it since each undirected cut can be directed as either $(S, \bar{S})$ or $(\bar{S}, S)$. □

Oblivious Routing using Electrical Flows

5.

Nothing helps a man to reform like thinking of the past with regret.

Fyodor Dostoyevsky, *The Idiot*

5.1. Introduction

Oblivious routing schemes use static-precomputed routing tables for selecting routing paths instead of routing paths that adapt dynamically to the observed traffic pattern of a parallel system. While at first glance this restriction seems like a serious barrier to obtaining good performance, it has been shown that in undirected networks oblivious routing does not only provide good theoretical guarantees [15, 58, 59], but is also an excellent choice for practical implementations [60, 61] due to its simple structure.

[15]: Räcke (2002), "Minimizing Congestion in General Networks"

[58]: Harrelson et al. (2003), "A Polynomial-time Tree Decomposition to Minimize Congestion"

[59]: Räcke (2008), "Optimal Hierarchical Decompositions for Congestion Minimization in Networks"

[60]: Applegate et al. (2003), "Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs"

[61]: Kumar et al. (2018), "Semi-Oblivious Traffic Engineering: The Road Not Taken"

[62]: Räcke et al. (2014), "Computing Cut-Based Hierarchical Decompositions in Almost Linear Time"

[63]: Englert et al. (2009), "Oblivious Routing for the Lp-norm"

[64]: Lawler et al. (2009), "Mixing Times and lp Bounds for Oblivious Routing"

[65]: Kelner et al. (2011), "Electric routing and concurrent flow cutting"

[66]: Schild et al. (2018), "Localization of Electrical Flows"

Formally, an oblivious routing scheme provides a unit flow $f_{s,t}$ between every source-target pair (s,t) in the network. When a demand $d_{s,t}$ between s and t appears, this unit flow is scaled by the demand to provide the required flow between source and target. When using an oblivious routing scheme for packet routing the path of a packet is chosen according to the flow so that the probability that the packet takes a certain edge is equal to the flow value along that edge. The main strength of any oblivious routing algorithm stems from the fact that determining the next hop for a packet can be done via a simple table lookup after precomputing the necessary routing tables.

In this chapter, we consider oblivious routing algorithms that aim to minimize network *congestion*, i.e., the maximum flow on any edge of the network. Most existing oblivious routing algorithms for this congestion cost-measure are *tree based* [15, 58, 59, 62, 63]. This means a convex combination of trees is embedded into the network. A routing path between two vertices s and t is then in principle chosen by first sampling a random tree and then following the path between s and t in this tree.

Another approach for oblivious routing is to use *electrical flows*. In an electrical flow routing one assigns a *resistance* (or its inverse, which is called *conductance*) to each edge of the graph. The flow between two vertices s and t is then defined by the current that would result when adding a voltage source between s and t. Lawler and Narayanan [64] studied the performance of electrical flows as an oblivious routing scheme. They show that if every edge is assigned unit resistance, then electrical routing has a *competitive ratio* of $\min\{\sqrt{m}, O(T_{\text{mix}})\}$ against any ℓ_p-norm *simultaneously*, where the competitive ratio is the ratio of the cost of routing any demand using oblivious routing to the cost of the optimal routing for that demand, and T_{mix} is the mixing time of a random walk on the graph. Kelner and Maymounkov [65] consider electrical routing on expander graphs (again with uniform resistances) and show that this scheme achieves small competitive ratio and is quite robust under edge deletions. Schild, Rao, and Srivastava [66] study the average length of an electrical flow, and as a consequence obtain $O(\log^2 n)$ competitive ratio for the special family of *edge-transitive* graphs. However, no non-trivial bound on the competitive ratio is known for electrical routing on general networks. This leads to the following important question:

Does there exist an electrical routing (with *non-uniform* conductances) that obtains a polylogarithmic competitive ratio with respect to the congestion cost-measure?

In this thesis, we give a partial answer to this question by showing how to obtain a competitive ratio of $O(\log^2 n)$ w.r.t. congestion with a *convex combination* of only $O(\sqrt{m})$ electrical flows[1].

1: In personal communication, Sidford and Lee [67] claimed that they had also observed that there exists an oblivious routing scheme based on a convex combination of $O(\sqrt{m})$ electrical flows that achieves $O(\log^2 n)$ competitive ratio.

[67]: Sidford et al. (2022), *Personal communication*

Theorem 5.1.1 *Given a capacitated graph $G = (V, E)$ with n vertices and m edges, there is an algorithm that finds an oblivious routing scheme composed of a convex combination of $O(\sqrt{m})$ electrical flows. The algorithm has competitive ratio $O(\log^2 n)$, runs in time $\widetilde{O}(m^{3/2})$, and can be implemented in parallel with $\widetilde{O}(\sqrt{m})$ depth.*

More specifically, we show that our competitive ratio is proportional to the best bound on the *localization of electrical flows* (Lemma 5.2.1), which is currently shown to be $O(\log^2 n)$. Hence, any improvement on the localization bound would lead to an improvement in our competitive ratio.

In contrast to this, all existing tree based oblivious routing schemes that guarantee a polylogarithmic competitive ratio require at least $\Omega(m)$ trees[2]. The best achievable competitive ratio with tree based schemes is $O(\log n)$. Note that electrical flows in particular generalize oblivious routings that are based on a convex combination of *spanning trees*[3] (as one could simply give all edges that are not part of the spanning tree a resistance of infinity). Therefore, our work shows that going from trees to electrical flows allows us to reduce the support of the convex combination from $O(m)$ to $O(\sqrt{m})$ with a slight increase in competitive ratio.

2: The routing schemes by Räcke [15] and Räcke et al. [62] are based on a single tree with different embeddings into the network; for this discussion, this is viewed as several trees.

3: In general tree based routings may embed trees that contain Steiner nodes, i.e., vertices that are not part of the network, and they may also use arbitrary paths to connect embedded vertices.

This reduction in the support of the convex combination also has an important implication for the construction time, and in particular for the parallel depth of the construction. The state of the art for constructing tree based oblivious routing schemes (with a polylogarithmic guarantee on the competitive ratio) is the multiplicative weights update method [68]. One iteration computes a low stretch spanning tree in time $\widetilde{O}(m)$ and updates weights/distances on edges. This results in depth $\widetilde{O}(m)$ and work $\widetilde{O}(m^2)$ for computing $O(m)$ trees. Räcke, Shah and Täubig [62] obtained a near-linear time algorithm for computing a single tree flow sparsifiers with quality $O(\log^4 n)$. While their construction can be adapted to an $O(\log^4 n)$ competitive oblivious routing scheme, the best-known efficient implementations of such a scheme require at least quadratic time in the size of the network.

[68]: Arora et al. (2012), "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications"

[62]: Räcke et al. (2014), "Computing Cut-Based Hierarchical Decompositions in Almost Linear Time"

We use the multiplicative weights update method with electrical flows. One iteration computes a set of resistances/conductances that is good on average for the current edge-weights and updates weights for the next iteration. We show that by using matrix sketching we can implement one iteration in time $\widetilde{O}(m)$ as in the tree case. Because electrical flows minimizes the ℓ_2-squared norm of the flow values, we can show that we only require $O(\sqrt{m})$ iterations to obtain a good oblivious routing scheme. This results in depth $\widetilde{O}(\sqrt{m})$ and work $\widetilde{O}(m^{3/2})$ for computing the convex combination of electrical flows.

Note that computing the routing tables for quickly determining the outgoing edge from the header of a packet adds additional work for tree-based and electrical-flow-based routing. For tree based routing this can be done with work $\widetilde{O}(n^2)$ per tree and depth $\widetilde{O}(1)$. For electrical flow based routing the work is $\widetilde{O}(mn)$ per flow with depth $\widetilde{O}(1)$. See Section 5.8.

We show that our techniques also apply to oblivious routing measured with respect to the ℓ_1-norm of loads on the edges, where we obtain a competitive ratio of $O(\log^2 n)$ with $O(\sqrt{m})$ electrical flows. See Section 5.7 for details.

5.1.1. Related Work

The study of oblivious routing was initiated by Valiant and Brebner [69], who developed an oblivious routing protocol for routing in the hypercube that routes any permutation in time that is only a logarithmic factor away from optimal. For the cost measure of minimizing congestion, Räcke [15] proved the existence of an oblivious routing scheme with polylogarithmic competitive ratio for any undirected network. This result was subsequently improved to a competitive ratio of $O(\log^2 n \log\log n)$ [58], and then to a competitive ratio of $O(\log n)$ [59], which is optimal. Englert and Räcke [63] extend these results to oblivious routing when the cost-measure is the ℓ_p-norm of the edge congestions.

Oblivious routing by electrical flows was first considered by Harsha et al. [70] for the goal of designing oblivious routing schemes that minimize the cost-measure $\|\cdot\|_2^2$, also known as *average latency*, for the case of a single target. In this scenario, using electrical flows is a very intuitive approach as it minimizes the $\|\cdot\|_2^2$ of the flow. Specifically, in their algorithm, every source in the oblivious routing scheme optimizes its flow as if no other source was active. They show that this gives a competitive ratio of $O(\log n)$ for the ℓ_2-norm squared of the flow.

Electrical flows also play a major role as a tool for speeding up flow computations (see e.g. [55, 71–77]). This is due to the fact that electrical flow computations reduce to solving Laplacian systems, which is a task that can be performed in nearly linear time. In addition, flow algorithms usually aim to minimize the ℓ_∞-norm in some way. The fact that the ℓ_2-norm is closer to the ℓ_∞ than e.g. the ℓ_1 norm helps in these optimizations. Electrical flows have also been used to generate alternative routes in road networks [78].

Ghaffari, Haeupler, and Zuzic [79] introduced the concept of *hop-constrained* oblivious routing. This is an oblivious routing scheme that is given an additional parameter h that can be viewed as an upper bound on the dilation used by an optimum routing. They show that one can obtain an oblivious routing scheme that guarantees a congestion of $\widetilde{O}(C_h)$ and a dilation of $\widetilde{O}(h)$, where C_h is the optimum congestion that can be obtained with routing paths of length at most h. However, their construction still suffers from the same drawback as tree-based routing schemes in that it requires $\Omega(m)$ iterations of multiplicative weights, which results in a depth of $\Omega(m)$. There has also been recent interest in computing competitive routing schemes with small support, as shown in the work of Zuzic, Haeupler, and Roeyskoe [80] in the context of *semi-oblivious* routing.

In the sub-polynomial competitive ratio regime, Kelner et al. [76] give an algorithm to compute an $n^{o(1)}$ competitive oblivious routing in $m^{1+o(1)}$ time, and use this as a subroutine to compute a $(1+\varepsilon)$-approximate maximum s-t flow and concurrent multicommodity flow in almost-linear time. Haeupler et al. [81] present a construction of routing tables for h-hop routing that runs in $O(D + \text{poly}(h))$ rounds of the CONGEST-model of distributed computing, and guarantee a competitive ratio of $n^{o(1)}$. This essentially means that the resulting packet routing algorithm can schedule any permutation in time $n^{o(1)}$ whenever this is possible, and it's extremely fast. However, the scheme crucially uses the fact that routing requests are initiated from both communication partners; it requires name-dependent routing (the node ids are changed during initialization to allow for compact routing tables), and it does not achieve a polylogarithmic competitive ratio.

[69]: Valiant et al. (1981), "Universal Schemes for Parallel Communication"

[55]: Gao et al. (2021), "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao"

[70]: Harsha et al. (2008), "Minimizing Average Latency in Oblivious Routing"

[71]: Madry (2016), "Computing Maximum Flow with Augmenting Electrical Flows"

[72]: Brand et al. (2022), "Faster Maxflow via Improved Dynamic Spectral Vertex Sparsifiers"

[73]: Dong et al. (2022), "Nested Dissection Meets IPMs: Planar Min-Cost Flow in Nearly-Linear Time"

[74]: Axiotis et al. (2020), "Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs"

[75]: Lee et al. (2013), "A New Approach to Computing Maximum Flows Using Electrical Flows"

[76]: Kelner et al. (2014), "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and Its Multicommodity Generalizations"

[77]: Christiano et al. (2011), "Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs"

[78]: Sinop et al. (2021), "Robust Routing Using Electrical Flows"

[79]: Ghaffari et al. (2021), "Hop-Constrained Oblivious Routing"

[80]: Zuzic et al. (2023), "Sparse Semi-Oblivious Routing: Few Random Paths Suffice"

[81]: Haeupler et al. (2022), "Hop-Constrained Expander Decompositions, Oblivious Routing, and Distributed Universal Optimality"

5.2. Preliminaries

For simplicity of presentation, we initially present the uncapacitated case, and explain the changes required for the capacitated case in Section 5.10.

5.2.1. Oblivious Routing

Graph. We will route on unweighted, undirected graphs. Choose an arbitrary orientation of the edges and let B be the $m \times n$ incidence matrix of G. Denote by b_e the row of B corresponding to edge e.

Demand. A demand $\chi \in \mathbb{R}^n$ encodes the flow deficit/surplus requirements for each vertex in the graph. A demand is *valid* if $\sum_i \chi_i = 0$, i.e., the required flow at all the vertices of the graph cancel out. Let $\mathscr{D} \subseteq \mathbb{R}^n$ denote the space of all valid demands. The vector b_e^T has unit demand across the endpoints of e (with the direction of demand decided by the orientation of the edge). If the demands are only between pairs of vertices (s,t), then we encode the demands between all pairs of vertices in a *demand vector* $d \in \mathbb{R}^{\binom{n}{2}}$.

Example 5.2.1 An (s,t)-flow is one where the demands of every vertex except s and t are zero.

Flow. A flow corresponding to a demand χ is a vector $f \in \mathbb{R}^m$ that ensures that the flow out of a vertex satisfies the demand, namely, for all $v \in V$,

$$\chi_v + \sum_{u \sim v} f_{uv} = 0.$$

While an oblivious routing scheme in general could be non-linear, we only consider linear routing schemes.

Oblivious Routing. An $m \times n$ matrix M that maps vertex demands to edge flows is called an *oblivious routing scheme* if for any valid demand χ the vertex demands created by the flow generated by M are equal to the input vertex demands. Formally, we require that $B^T M$ is identity on $\mathscr{D}$, i.e., $B^T M \chi = \chi$ for all valid demands $\chi \in \mathscr{D}$.

4: Since each demand is routed separately, two flows that travel in opposite directions on an edge do not cancel out.

Competitive Ratio. For an oblivious routing M and any collection of demands $D = \{\chi_i\}_i$, the flow across edge e for routing demand χ_i is given by $|b_e M \chi_i|$. Thus the total flow (without cancellations)[4] across edge e when routing all the demands in D *independently* using M is given by $f_e = \sum_{\chi_i \in D} |b_e M \chi_i|$. We would like to minimize the ℓ_p norm (for us, $p \in \{1, \infty\}$) of the flow resulting from routing these demands independently with an oblivious routing M, as compared with the optimal routing that can adaptively choose the routing based on the demands. If $OPT_p(D)$ is the cost resulting from the optimal routing, and f is the flow described above, then the competitive ratio of the oblivious routing scheme M is given by

$$\beta_p(M) = \max_D \frac{\|f\|_p}{OPT_p(D)}$$

Kelner and Maymounkov [65, Theorem 3.1] show that for a linear oblivious routing on an uncapacitated graph, the worst case demand set is routing one unit of flow across every edge of the graph, i.e., the worst case requests are the columns of B^T. Since the optimal routing is to route each demand across the same edge, $OPT(B^T) = 1$. For $p \in \{1, \infty\}$, the competitive ratio is then given by $\beta_p(M) = \|MB^T\|_p$.

Load. For oblivious routing M, the flow across edge e for the demand b_f is given by $\text{load}_M(f \to e)$. This can be obtained from the matrix M as $\text{load}_M(f \to e) = |(Mb_f^\mathsf{T})_e|$. The sum of all flows across an edge e for every possible edge demand vector b_f is then the load across the edge e.

$$\text{load}_M(e) = \sum_f \text{load}_M(f \to e) = \sum_f |(Mb_f^\mathsf{T})_e|.$$

The *congestion*, or the competitive ratio for the ℓ_∞ norm, is thus the maximum load on any edge in the graph (by definition of $\|MB^\mathsf{T}\|_\infty$).

Convexity of Load. Load is convex on oblivious routings. For two oblivious routings M_0 and M_1, let $M_\lambda = \lambda M_1 + (1-\lambda)M_0$ for $\lambda \in (0,1)$. Then

$$\begin{aligned}\text{load}_{M_\lambda}(f \to e) &= |((\lambda M_1 + (1-\lambda)M_0)b_f^\mathsf{T})_e| \le \lambda|(M_1 b_f^\mathsf{T})_e| + (1-\lambda)|(M_0 b_f^\mathsf{T})_e| \\ &= \lambda \text{load}_{M_1}(f \to e) + (1-\lambda)\text{load}_{M_0}(f \to e)\end{aligned}$$

The Simplex. We use $\Delta^m = \{x \in \mathbb{R}^m : \sum x_i = 1, x_i \ge 0\}$ to denote the m-simplex, and any $p \in \Delta^m$ is a probability distribution over the edges.

5.2.2. Electrical Flow

Graph. Let (G, w) be a weighted, undirected graph with edge weights $\{w_e\}$. Choose an arbitrary orientation of the edges. B is the $m \times n$ incidence matrix of G. If $W = \text{diag}(w)$ is the $m \times m$ diagonal matrix of edge weights, then the $n \times n$ *Laplacian matrix* is given by $L = B^\mathsf{T}WB$, and $L^\dagger$ is its pseudoinverse. Notationally, by $L^\dagger$ we always mean the pseudoinverse of the Laplacian with respect to the weighted graph and *not* the underlying unweighted graph.

Think of these weights as the conductances of the edges in an electrical network.

Electrical Flow. Given a weighted graph (G, w) and a demand χ, the electrical flow corresponding to χ is given by $WBL^\dagger\chi$.

This corresponds exactly to routing the demand using resistances w^{-1} on the edges and voltages $L^\dagger\chi$ at the vertices.

Load. By the above characterization of electrical flow, the load across an edge e for unit current demand across f is given by $\text{load}_w(f \to e) = w_e|b_e L^\dagger b_f^\mathsf{T}|$. The load on an edge is the sum of all loads on the edge for demands b_f for all edges f, given by $\text{load}_w(e) = w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|$.

Weights Determine Electrical Flow. An electrical flow is completely determined by the edge weights. That is, given an edge weighting w of an unweighted graph G, there is a unique routing $WBL^\dagger$ corresponding to this network. If the unweighted graph G is fixed, we denote the load due to weights w on an edge e as $\text{load}_w(e)$.

Electrical Flows are Oblivious. Let G be an unweighted graph for which we wish to find an oblivious routing. Then for any weighting w of the graph, the corresponding electrical flow $M_w = WBL^\dagger$ is an oblivious routing. This is easy to check, since $B^\mathsf{T}WBL^\dagger = (B^\mathsf{T}WB)(B^\mathsf{T}WB)^\dagger$. We will refer to M_w as the *electrical routing with respect to the weights* $\{w_e\}$.

We can then discuss interchangeably the weights and its corresponding electrical oblivious routing.

Localization. We will repeatedly use the following so-called *localization lemma*. A concrete instantiation with $\alpha_{\text{LOCAL}} = c\log^2(n)$ for some constant c was given by Schild et al. [66].

The bound on average electric flow path lengths is obtained from this lemma using the Perron-Frobenius theorem.

Lemma 5.2.1 (Localization [66]) *Let G be a graph with weights $\{w_e\}$. Then for any vector $\ell \in \mathbb{R}^m_{\geq 0}$,*

$$\sum_{e,f \in E} \ell_e \ell_f \sqrt{w_e w_f} |b_e L_G^{\dagger} b_f^{\mathsf{T}}| \leq \alpha_{LOCAL} \cdot \|\ell\|_2^2$$

5.2.3. Matrix Sketches

To speed up our algorithms we use a sketching result by Indyk [82, Theorem 3], adapted from a formulation by Schild [83, Theorem 9.13].

[82]: Indyk (2006), "Stable distributions, pseudorandom generators, embeddings, and data stream computation"

[83]: Schild (2018), "An almost-linear time algorithm for uniform random spanning tree generation"

This sketch matrix will allow us to estimate the loads on all edges much cheaper than estimating all of them individually.

Theorem 5.2.2 ([82]) *Given $m \in \mathbb{Z}_{\geq 1}$, $\delta \in (0,1)$, and $\varepsilon \in (0,1)$, there is a sketch matrix $C = \textsc{SketchMatrix}(m, \delta, \varepsilon) \in \mathbb{R}^{\ell \times m}$ and an algorithm $\textsc{RecoverNorm}(s)$ for $s \in \mathbb{R}^{\ell}$ such that the following properties hold:*

- *(Approximation) For any $v \in \mathbb{R}^m$, with probability at least $1-\delta$ over the randomness of $\textsc{SketchMatrix}$, the value of $r = \textsc{RecoverNorm}(Cv)$ is*
$$(1-\varepsilon)\|v\|_1 \leq r \leq (1+\varepsilon)\|v\|_1$$
- *$\ell = c/\varepsilon^2 \cdot \log(1/\delta)$ for some constant $c > 1$*
- *(running time) $\textsc{SketchMatrix}$ and $\textsc{RecoverNorm}$ take time $O(\ell m)$ and $\operatorname{poly}(\ell)$ respectively.*

We will require that our theorems hold with high probability, and thus we will set δ to be $1/\operatorname{poly}(n)$. We will also set the approximation constant ε to be $1/2$, which gives $\ell = O(\log n)$.

5.3. Technical Overview

For so-called *linear oblivious routing schemes* (such as tree-based routing schemes or electrical flows) it is well known that the worst demand is a demand of 1 along every edge in the unweighted network. An optimal algorithm can trivially route this demand by sending 1 unit of flow along every edge in the network, and, hence, has maximum congestion 1. Let $\operatorname{load}_w(e)$ denote the load on edge e generated by an electrical flow routing (with conductances w) when routing this worst case demand. The competitive ratio of the routing scheme is then $\max_e \operatorname{load}_w(e)$. We can search for a good convex combination of electrical flow routings w_i with a linear program:

The columns of this LP correspond to all possible weightings, but since multiplicative weights update only returns a single set of weights in each round, the support is bounded by the number of iterations.

$$\begin{array}{lll} \text{minimize} & \alpha & \\ \text{such that} & \forall e \quad \sum_i \lambda_i \cdot \operatorname{load}_{w_i}(e) & \leq \alpha \\ & \sum_i \lambda_i & = 1 \\ & \forall i \quad \lambda_i & \geq 0 \ . \end{array}$$

We want to find a good solution to this problem, say with objective value β. For this the multiplicative weights update method maintains a set of weights $p_e \geq 0$ on the edges with $\sum_e p_e = 1$. In each iteration it computes *one* routing R_i such that R_i fulfills $\sum_e p_e \operatorname{load}_{R_i}(e) \leq \beta$, i.e., the weighted total load over all edges for R_i is at most β. Here $\operatorname{load}_{R_i}(e)$ denotes the load induced on edge e when routing a demand of 1 along every edge using the routing R_i. This means *on average* the edges have load at most β when using routing R_i. For the next iteration the weight p_e of edges with $\operatorname{load}_{R_i}(e) > \beta$ is increased

while edges with $\text{load}_{R_i}(e) < \beta$ decrease their weight. In the end the convex combination of all the routings will have load at most β for every edge.

To apply this scheme we first need a bound β such that we always can (efficiently) find a routing with $\sum_e p_e \text{load}_{R_i}(e) \leq \beta$. For tree-based routing one can use small stretch trees [84, 85] to solve this problem with $\beta = O(\log n)$. For electrical routing we show that we can bound β using a specific choice of parameters in the so-called *localization lemma* due to Schild et al. [66]. Informally, the localization lemma states that the average length of flow paths in an electrical routing is small. In particular, we show that we can find parameters for localization which guarantee the existence of conductances w such that the load when routing via the electrical flow with conductances w (say load_w) fulfills $\sum_e p_e \text{load}_w(e) \leq \beta$, with $\beta = O(\log^2 n)$.

[84]: Fakcharoenphol et al. (2003), "A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics"

[85]: Abraham et al. (2008), "Nearly Tight Low Stretch Spanning Trees"

Secondly we need to be able to efficiently compute the load on every edge after computing the routing R_i in an iteration, in order to be able to adjust the weights for the next iteration. Naïvely routing across each edge using R_i would require time $O(m^2)$ for solving m Laplacian systems to compute the electrical flow between the end points of each edge. We use a sketching result due to Indyk [82] for approximating the ℓ_1-norm of vectors. This allows us to approximate the loads due to all m electrical flows in time $\widetilde{O}(m)$. We show that this approximation is still sufficient for the multiplicative weights method to work correctly.

The advantage of using electrical flows within the multiplicative weights update framework is that it allows to derive a better bound on the convergence time. The number of iterations required for the method is proportional to how far each inequality, namely $\text{load}_w(e) \leq \beta$ for each e, is violated by the routing found in each iteration. This is the so-called *width* of the algorithm, and we bound the width by $O(\sqrt{m})$ by regularizing the weights to be nearly uniform. This in turn implies that we require only $O(\sqrt{m})$ iterations.

5.4. Algorithm for routing

We give an algorithm that returns a routing which achieves competitive ratio $O(\alpha_{\text{LOCAL}})$ by taking a convex combination of $\sqrt{m}$ electrical flows, where α_{LOCAL} is the best localization result for electrical flows in Lemma 5.2.1. We use the *multiplicative weights update* method (MWU) to obtain this guarantee. The algorithm is presented in Algorithm 5.1 (ComputeRouting).

For the linear program we are solving with MWU, the primal asks for a convex combination of electrical routings that gives low competitive ratio, where the coefficients of the convex combination are collected in λ. The dual of this linear program is equivalent to solving

$$\begin{aligned} \text{minimize} \quad & \alpha \\ \text{such that} \quad & \forall e \ \textstyle\sum_i \lambda_i \cdot \text{load}_{w_i}(e) \leq \alpha \\ & \textstyle\sum_i \lambda_i = 1 \\ & \forall i \qquad \lambda_i \geq 0 \ . \end{aligned}$$

$$\max_{p \in \Delta^m} \min_i \sum_e p_e \text{load}_{w_i}(e),$$

Denote the optimal values of the primal and dual by α^* and β^* respectively. Since $\alpha^* = \beta^*$ by strong duality, we can obtain an existential bound on α^* by showing that $\beta^* \leq \alpha_{\text{LOCAL}}$. This bound on β^* is obtained by bounding the equation above using the weighting returned by Lemma 5.5.1 which returns, for any $p \in \Delta^m$, a weighting w for which the p_e weighted average load is smaller than $2\alpha_{\text{LOCAL}}$. The proof of the bound in Lemma 5.5.1 uses localization, and we wish to convert this existential result into an algorithmic one, which we do using MWU in Algorithm 5.1.

Algorithm 5.1: ComputeRouting, to achieve $O(\alpha_{\text{LOCAL}})$-competitive oblivious routing.

Input: An unweighted graph G.
Output: An oblivious routing scheme on G.

Set $\rho \leftarrow \sqrt{2m}$ and $T \leftarrow \frac{8\rho \ln m}{\alpha_{\text{LOCAL}}}$
Initialize $x_e^{(0)} \leftarrow 1$ for all $e \in E$, and $X^{(0)} \leftarrow m$
for $t = 1, 2, \ldots, T$ **do**
  Set $p_e^{(t)} \leftarrow x_e^{(t-1)} / X^{(t-1)}$ for all $e \in E$
  Set $w_e^{(t)} \leftarrow (p_e + 1/m)^{-1}$ for all $e \in E$
  Set $W^{(t)} \leftarrow \text{diag}(w)$
  Set $M^{(t)} \leftarrow W^{(t)} B \left(B^\intercal W^{(t)} B\right)^\dagger$
  Set $\text{aload}_{w^{(t)}} \leftarrow$ GetApproxLoad$(G, w, 1/2)$
  Set $x_e^{(t)} \leftarrow x_e^{(t-1)} \cdot \left[1 + \frac{1}{2\rho} \cdot \text{aload}_{w^{(t)}}(e)\right]$
  Set $X^{(t)} \leftarrow \sum_e x_e^{(t)}$
end
return $M^* = \frac{1}{T} \cdot \sum_{t=1}^{T} M^{(t)}$

Algorithm 5.2: GetApproxLoad, to compute approximate loads for electrical routing.

Input: A graph G, weights $\{w_e\}_{e \in E}$ on the edges, approximation factor ε.
Output: Approximation $\{\text{aload}_w(e)\}_{e \in E}$ to the load on the edges.

Let B be the edge-vertex incidence matrix of G
Let $L := B^\intercal \text{diag}(w) B$ be the Laplacian matrix
Set $C \leftarrow$ SketchMatrix$(m, n^{-10}, \varepsilon)$
Set $X \leftarrow B^\intercal C^\intercal$
Let $X^{(i)}$ be the i^{th} column of X for all $i \in [\ell]$
Set $U^{(i)} \leftarrow$ LapSolve$(L, X^{(i)})$ for all $i \in [\ell]$
Set $U \leftarrow (U^{(1)}, U^{(2)}, \ldots, U^{(t)})$ ▷ $U = (CBL^\dagger)^\intercal$
Set $\text{aload}_w(e) \leftarrow w_e \cdot$ RecoverNorm$(U^\intercal b_e)$ for all $e \in E$
return aload_w

Our MWU algorithm can be described as the following primal-dual algorithm running for T iterations: We initially start with the primal vector $\lambda = 0$ and dual vector $p_e^{(1)} = 1/m$. We then build a primal solution incrementally as follows. At iteration t, we look at the dual variables $p_e^{(t)}$ of the edges, and find a particular set of weights $w_e^{(t)}$ such that routing with respect to these weights gives low average load with respect to the dual variables, i.e., $\sum_e p_e^{(t)} \text{load}_{w^{(t)}}(e)$ is low. As in the existential case, we show a bound of $O(\alpha_{\text{LOCAL}})$ on this average load using localization.

Recall that the primal vector λ gives the coefficients for a convex combination of electrical routings. We now increase the primal coefficient corresponding to the routing that was found at this iteration, $\lambda_{M^{(t)}}$, by $1/T$. We essentially "add" this routing to the final convex combination that we output at the end of the algorithm. Now we update the dual variables for the next iteration. For the routing returned in the next iteration, we want to reduce the load on edges that had high load in the current iteration. To this end, we compute the loads induced on each edge (using GetApproxLoad), and adjust the dual variables based on how high the loads for the current routing $M^{(t)}$ are.

At the end of T iterations, λ is the uniform combination of all T electrical routings computed during each iteration of the algorithm. The analysis of MWU shows that our dual variable updates ensure low load on all the edges for the returned routing. The number of iterations T is proportional to how far each primal inequality, namely $\text{load}_{w^{(t)}}(e) \leq \alpha_{\text{LOCAL}}$ for each e, is violated

by the routing found in each iteration. This is the *width* of the algorithm, and we bound the width by $O(\sqrt{m})$.

For the running time, note that evaluating loads for all the edges naïvely would take time $O(m^2)$ in each iteration, since it would involve calculating $|b_e L^\dagger b_f^\mathsf{T}|$ for every pair of edges $(e, f) \in E^2$. However, the load on each edge can also be expressed as the ℓ_1 norm of the vector $BL^\dagger b_e$. We use a sketching result by Indyk [82] stated in Theorem 5.2.2, which gives a sketch matrix C that preserves ℓ_1 norms up to small multiplicative errors. Similar ideas have been implicitly used by Schild [83], and later in other works [86, 87] as well. Our sketching improves the running time from $O(m^2)$ to $\widetilde{O}(m)$ per iteration. Since there are $T = O(\sqrt{m})$ iterations, we get a bound of $\widetilde{O}(m\sqrt{m})$ for the total running time to construct our routing.

[86]: Li et al. (2018), "Spectral Subspace Sparsification"

[87]: Forster et al. (2021), "Minor Sparsifiers and the Distributed Laplacian Paradigm"

For simplicity, we first assume that in GetApproxLoad we have an exact Laplacian solver that runs in time $\widetilde{O}(m)$, and present below the analysis in Section 5.5. The analysis when using an approximate Laplacian solver is more technical, and we present it in Section 5.9. The complication to overcome when using an approximate Laplacian solver in line 6 of GetApproxLoad is the following: The input to RecoverNorm might not necessarily be of the form Cv for some $v \in \mathbb{R}^m$, since we use the approximate Laplacian solver on $B^\mathsf{T}C^\mathsf{T}$, and thus would have obtained $\textsc{approx}(CBL^\dagger)b_e$ at the end. Note that if we instead had $C \cdot \textsc{approx}(BL^\dagger b_e)$ this would be fine, since we would get that $\textsc{RecoverNorm}(CBL^\dagger b_e) \approx \|BL^\dagger b_e\|_1$ and $\|BL^\dagger b_e\|_1 \approx \|\textsc{approx}(BL^\dagger b_e)\|_1$, and the resulting guarantee would be the product of these two guarantees.

At a high level, we have an approximation to the vector *after* sketching ($\textsc{approx}(Cv)$) instead of an approximation to the vector *before* sketching ($C \cdot \textsc{approx}(v)$). Since RecoverNorm is not a norm, we do not have a guarantee that it behaves well on inputs that are close to each other. We need to argue that RecoverNorm still returns a useful answer when given as input a vector that does not belong to the column space of C, but is nevertheless close in the L norm to the required vector Cv. This requires some technical analysis.

5.5. Proof of correctness

The proof is an adaptation of the multiplicative weight update proof to our setting. The proof uses the following three lemmas. We defer their proofs to Sections 5.5.1, 5.5.2, and 5.5.3 respectively.

Lemma 5.5.1 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \le 2\alpha_{LOCAL}$.*

The *(p_e weighted) average edge load* is $O(\alpha_{\text{LOCAL}})$.

Lemma 5.5.2 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\text{load}_w(e) \le \sqrt{2m}$ for every edge e.*

In each iteration of ComputeRouting, the true loads on each edge are $O(\sqrt{m})$ away from α_{LOCAL}.

Lemma 5.5.3 *For any approximation factor $0 < \varepsilon < 1$, and any weighted graph (G, w), let $\text{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\text{aload}_w = \textsc{GetApproxLoad}(G, w, \varepsilon)$ be the approximate loads computed*

The error introduced by using a matrix sketch in GetApproxLoad is not too large, i.e., that GetApproxLoad approximates the true loads well.

by the algorithm. Then with probability $\geq 1 - 1/\text{poly}(n)$,

$$(1-\varepsilon)\cdot \text{load}_w(e) \leq \text{aload}_w(e) \leq (1+\varepsilon)\cdot \text{load}_w(e) \qquad \text{for all } e \in E$$

For simplicity, we assume that the Laplacian solver used in GetApproxLoad is exact and runs in time $\widetilde{O}(m)$. The analysis with an approximate solver is given in Section 5.9. We now track the potential function $\|x^{(t)}\|_1$.

Upper bound on total increase in potential.

Lemma 5.5.4 *For any* $t \geq 1$, $\|x^{(t)}\|_1 \leq \|x^{(t-1)}\|_1 \cdot \exp(2\alpha_{LOCAL}/\rho)$. *At the end of the algorithm,* $\|x^{(T)}\|_1 \leq m \cdot \exp(2\alpha_{LOCAL}T/\rho)$.

Proof. The implication for $\|x^{(T)}\|_1$ follows from the former statement and noting that $\|x^{(0)}\|_1 = m$. For any $t \geq 1$, we have

$$\begin{aligned}
\|x^{(t)}\|_1 &= \sum_e x_e^{(t)} = \sum_e x_e^{(t-1)} \cdot \left(1 + \frac{1}{2\rho}\cdot \text{aload}_{w^{(t)}}(e)\right) \\
&= \sum_e x_e^{(t-1)} + \frac{1}{2\rho}\cdot \sum_e x_e^{(t-1)} \cdot \text{aload}_{w^{(t)}}(e) \\
&\leq \sum_e x_e^{(t-1)} + \frac{\|x^{(t-1)}\|_1}{\rho}\cdot \sum_e \frac{x_e^{(t-1)}}{\|x^{(t-1)}\|_1}\cdot \text{load}_{w^{(t)}}(e) && \text{(by Lemma 5.5.3)} \\
&\leq \|x^{(t-1)}\|_1 + \frac{\|x^{(t-1)}\|_1}{\rho}\cdot 2\alpha_{\text{LOCAL}} && \text{(by Lemma 5.5.1)} \\
&\leq \|x^{(t-1)}\|_1 \cdot \exp\left(\frac{2\alpha_{\text{LOCAL}}}{\rho}\right).
\end{aligned}$$

□

Lower bound on potential of an edge.

Lemma 5.5.5 *Let* M^* *be the routing returned by the algorithm. For any edge* e,

$$x_e^{(T)} \geq \exp\left(\frac{T}{8\rho}\cdot \text{load}_{M^*}(e)\right).$$

Proof. For any edge e and $t \geq 1$, we have

$$\begin{aligned}
x_e^{(t)} &= \prod_{t'=1}^{t}\left(1 + \frac{1}{2\rho}\cdot \text{aload}_{w^{(t')}}(e)\right) \\
&\geq \prod_{t'=1}^{t}\left(1 + \frac{1}{4\rho}\cdot \text{load}_{w^{(t')}}(e)\right) && \text{(by Lemma 5.5.3)} \\
&\geq \prod_{t'=1}^{t}\exp\left(\frac{1}{8\rho}\cdot \text{load}_{w^{(t')}}(e)\right) && (e^x \leq 1+2x \text{ for } 0<x<1, \text{ and Lemma 5.5.2}) \\
&= \exp\left(\frac{1}{8\rho}\cdot \sum_{t'=1}^{t}\text{load}_{w^{(t')}}(e)\right)
\end{aligned}$$

In particular, for $t = T$, we have by convexity of load that

$$\begin{aligned}
x_e^{(T)} &\geq \exp\left(\frac{1}{8\rho}\cdot \sum_{t'=1}^{T}\text{load}_{w^{(t')}}(e)\right) \\
&\geq \exp\left(\frac{T}{8\rho}\cdot \text{load}_{M^*}(e)\right).
\end{aligned}$$

□

Theorem 5.5.6 *The routing returned by Algorithm 5.1 has competitive ratio $O(\alpha_{LOCAL})$.*

Proof. Combining Lemmas 5.5.4 and 5.5.5, for any edge e,

$$m \exp\left(\frac{2\alpha_{\text{LOCAL}} T}{\rho}\right) \geq \|x^{(T)}\|_1 \geq x_e^{(T)} \geq \exp\left(\frac{T}{8\rho} \cdot \text{load}_{M^*}(e)\right)$$

which in particular gives the required upper bound on $\text{load}_{M^*}(e)$ for any edge e, since

$$\begin{aligned} \text{load}_{M^*}(e) &\leq 16\alpha_{\text{LOCAL}} + \frac{8\rho \log m}{T} \\ &= O(\alpha_{\text{LOCAL}}). \end{aligned}$$

□

5.5.1. Bound on Average Loads

Lemma 5.5.1 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \leq 2\alpha_{LOCAL}$.*

Proof. Setting ℓ_e to $1/\sqrt{w_e}$ and applying Lemma 5.2.1, we get

$$\begin{aligned} \sum_e p_e \text{load}_w(e) &= \sum_e p_e \sum_f \text{load}_w(f \to e) \\ &= \sum_e p_e \cdot w_e \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of load)} \\ &= \sum_e p_e \cdot (p_e + 1/m)^{-1} \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } w_e\text{)} \\ &\leq \sum_e \sum_f |b_e L^\dagger b_f^\mathsf{T}| \\ &= \sum_{e,f} 1/\sqrt{w_e w_f} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\ &= \sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by choice of } \ell_e\text{)} \\ &\leq \alpha_{\text{LOCAL}} \cdot \|\ell\|_2^2 && \text{(by Lemma 5.2.1)} \\ &= \alpha_{\text{LOCAL}} \cdot \sum_e 1/w_e \\ &= \alpha_{\text{LOCAL}} \cdot \sum_e (p_e + 1/m) \\ &= 2\alpha_{\text{LOCAL}}. \end{aligned}$$

□

5.5.2. Bounding the Width

Next we show that the width is bounded above by $O(\sqrt{m})$. We first prove a couple of properties of the Π matrix that we will use in our analysis.

Lemma 5.5.7 *The matrix Π is a projection matrix.*

Proof. We show that $\Pi^2 = \Pi$.

$$\begin{aligned}\Pi^2 &= \left(W^{1/2}BL^\dagger B^\mathsf{T}W^{1/2}\right)\cdot\left(W^{1/2}BL^\dagger B^\mathsf{T}W^{1/2}\right)\\ &= \left(W^{1/2}B\right)\cdot\left(L^\dagger B^\mathsf{T}WBL^\dagger\right)\cdot\left(B^\mathsf{T}W^{1/2}\right)\\ &= \left(W^{1/2}B\right)\cdot L^\dagger\cdot\left(B^\mathsf{T}W^{1/2}\right) = \Pi.\end{aligned}$$

□

This lemma is used to bound the width for both the ℓ_∞ and the ℓ_1 case.

Lemma 5.5.8 *Let G be a graph with weights $\{w_e\}$ and let L be the Laplacian matrix associated with G. For any edge e, we have that*

$$\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \le 1.$$

Proof. By Lemma 5.5.7 we know that Π is a projection matrix. This in particular implies that the diagonal entries of Π (and hence Π^2) are less then 1. Thus,

$$\begin{aligned}\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 &= \sum_f \left(\sqrt{w_e w_f}\cdot|b_e L^\dagger b_f^\mathsf{T}|\right)^2\\ &= \sum_f \Pi(e,f)^2 = \Pi^2(e,e) \le 1.\end{aligned}$$

□

Lemma 5.5.2 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)^{-1}$ satisfies $\mathrm{load}_w(e) \le \sqrt{2m}$ for every edge e.*

Proof. Fixing an edge e,

$$\begin{aligned}\mathrm{load}_w(e) = w_e\sum_f |b_e L^\dagger b_f^\mathsf{T}| &\le \sum_f \sqrt{w_e/w_f}\cdot\sqrt{w_e w_f}|b_e L^\dagger b_f^\mathsf{T}|\\ &\le \sqrt{\sum_f w_e/w_f}\cdot\sqrt{\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2}\end{aligned}$$

(by Cauchy-Schwarz)

We now bound each of the two terms separately. For the first term, note that

$$w_e\cdot\sum_f 1/w_f = (p_e + 1/m)^{-1}\cdot\sum_f\left(p_f + 1/m\right) \le 2\cdot(p_e + 1/m)^{-1} \le 2\cdot 1/(1/m) = 2m.$$

For the second term, by Lemma 5.5.8, we know that

$$\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \le 1.$$

Putting these two inequalities together, we get that $\mathrm{load}_w(e) \le \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. □

5.5.3. Proof of GetApproxLoad correctness assuming exact Laplacian solver

We use the guarantees of SketchMatrix and RecoverNorm provided by Theorem 5.2.2 to prove the following lemma.

Lemma 5.5.3 *For any approximation factor $0 < \varepsilon < 1$, and any weighted graph (G, w), let $\text{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\text{aload}_w = \textsc{GetApproxLoad}(G, w, \varepsilon)$ be the approximate loads computed by the algorithm. Then with probability $\geq 1 - 1/\text{poly}(n)$,*

$$(1-\varepsilon) \cdot \text{load}_w(e) \leq \text{aload}_w(e) \leq (1+\varepsilon) \cdot \text{load}_w(e) \qquad \text{for all } e \in E$$

Proof. Note that GetApproxLoad sends $(L^\dagger B^\mathsf{T} C^\mathsf{T})^\mathsf{T} b_e$ to RecoverNorm. This simplifies to $CBL^\dagger b_e$. We have

$$(1-\varepsilon) \cdot \|BL^\dagger b_e\|_1 \leq \textsc{RecoverNorm}(BL^\dagger b_e) \leq (1+\varepsilon) \cdot \|BL^\dagger b_e\|_1 \qquad \text{(by Theorem 5.2.2)}$$

Since $\text{load}_w(e) = w_e \cdot \|BL^\dagger b_e\|_1$, multiplying the above inequality by w_e,

$$(1-\varepsilon) \cdot \text{load}_w(e) \leq \textsc{GetApproxLoad}(G, w) \leq (1+\varepsilon) \cdot \text{load}_w(e). \qquad \square$$

5.6. Running time analysis

In this section, we show that Algorithm 5.1 (ComputeRouting) runs in time $\widetilde{O}(m^{3/2})$. We show that each iteration of the `for` loop in ComputeRouting takes time $\widetilde{O}(m)$, and that $T = O(\sqrt{m})$, which gives the claim. We first show that Algorithm 5.2 (GetApproxLoad) runs in time $\widetilde{O}(m)$.

Lemma 5.6.1 *Algorithm 5.2 runs in time $\widetilde{O}(m)$.*

Proof. Calculating B and L takes time $O(m)$. By Theorem 5.2.2, building the sketch matrix C takes time $O(\ell m)$. Since we set $\delta = n^{-10}$ and $\varepsilon = 1/2$ for the sketch matrix, we get $\ell = O(\log n)$, which gives $O(\ell m) = \widetilde{O}(m)$.

Each row of $B^\mathsf{T} C^\mathsf{T}$ can be obtained as follows: Since the row of B^T corresponding to vertex u has $\deg(u)$ non-zero entries, the row of $B^\mathsf{T} C^\mathsf{T}$ corresponding to vertex u can be obtained by taking the sum of every row of C^T corresponding to an edge that is incident to u. Since each row of C^T has ℓ entries, this involves $O(\deg(u) \cdot \ell)$ calculations for computing row u of $B^\mathsf{T} C^\mathsf{T}$. Since $\sum_u \deg(u) = 2m$, this gives an overall bound of $\widetilde{O}(m)$ for calculating $B^\mathsf{T} C^\mathsf{T}$. Each Laplacian solver takes time $\widetilde{O}(m)$, and since we solve for ℓ vectors, all Laplacian solves together take time $\widetilde{O}(m)$ as well.

Finally, we perform RecoverNorm for m edges. Each invocation of RecoverNorm takes time $O(\ell)$ by Theorem 5.2.2, and thus all calls to RecoverNorm together run in time $\widetilde{O}(m)$. This proves the lemma. $\square$

We use this to show that each iteration of the `for` loop in ComputeRouting runs in time $\widetilde{O}(m)$, and thus the entire algorithm runs in time $\widetilde{O}(m^{3/2})$.

Lemma 5.6.2 *Algorithm 5.1 runs in time $\widetilde{O}(m^{3/2})$.*

Proof. For each iteration of the for loop, normalizing the $x_e^{(t-1)}$s and calculating $w_e^{(t)}$ (and thus $W^{(t)}$) takes time $O(m)$. Calculating approximate loads is $\widetilde{O}(m)$ by Lemma 5.6.1. Since computing $x_e^{(t)}$ takes time $O(m)$, each iteration runs in time $\widetilde{O}(m)$. Thus the algorithm runs in time $\widetilde{O}(Tm) = \widetilde{O}(m^{3/2})$ by choice of $T = O(\sqrt{m})$. $\square$

5.7. Extension to ℓ_1 norm Oblivious Routing

In this section we prove that a convex combination of $O(\sqrt{m})$ electrical routings gives an oblivious routing scheme with respect to the ℓ_1 norm with $O(\log^2 n)$ competitive ratio, by bounding the maximum *stretch* of an edge.

Consider the oblivious routing operator $M_w = WBL^\dagger$. The *stretch* of an edge $e \in E$ with respect to the routing operator M_w is given by:

$$\text{stretch}_w(e) := \sum_f w_f |b_e L^\dagger b_f^T|.$$

Similar to earlier, our goal is to solve the following linear program.

$$\begin{array}{lll} \min & \alpha & \\ \text{s.t.} & \forall e & \sum_i \lambda_i \cdot \text{stretch}_{w_i}(e) \leq \alpha \\ & & \sum_i \lambda_i = 1 \\ & \forall i & \lambda_i \geq 0. \end{array}$$

With the same dual construction as for load, the dual is equivalent to

$$\max_{p \in \Delta^m} \min_i \sum_e p_e \text{stretch}_{w_i}(e), \tag{5.1}$$

We bound the average stretch and the width for stretch.

This lemma gives a weighting $w(p)$ such that electrical routing on G with weights $w(p)$ gives low average stretch.

Lemma 5.7.1 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)$ satisfies $\sum_e p_e \text{stretch}_w(e) \leq 2\alpha_{LOCAL}$.*

Proof. Setting ℓ_e to $\sqrt{w_e}$ and applying Lemma 5.2.1, we get

$$\begin{aligned} \sum_e p_e \text{stretch}_w(e) &= \sum_e p_e \cdot \sum_f w_f \cdot \text{stretch}_w(e \to f) \\ \text{(by definition of stretch)} \quad &= \sum_e p_e \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^T| \\ &\leq \sum_e (p_e + 1/m) \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^T| \\ \text{(by definition of } w_e) \quad &\leq \sum_e w_e \cdot \sum_f w_f \cdot |b_e L^\dagger b_f^T| \\ &= \sum_e \sum_f w_e \cdot w_f \cdot |b_e L^\dagger b_f^T| \\ &= \sum_{e,f} \sqrt{w_e w_f} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^T| \\ \text{(by choice of } \ell_e) \quad &= \sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^T| \\ \text{(by Lemma 5.2.1)} \quad &\leq \alpha_{\text{LOCAL}} \cdot \|\ell\|_2^2 \\ &= \alpha_{\text{LOCAL}} \cdot \sum_e w_e \\ &= \alpha_{\text{LOCAL}} \cdot \sum_e (p_e + 1/m) \\ &= 2\alpha_{\text{LOCAL}}. \end{aligned}$$

□

Lemma 5.7.2 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = (p_e + 1/m)$ satisfies* $\text{stretch}_w(e) \leq \sqrt{2m}$ *for every edge e.*

We bound the stretch as in the ℓ_∞ case.

Proof. Fixing an edge e,

$$\begin{aligned}\text{stretch}_w(e) &= \sum_f w_f |b_e L^\dagger b_f^\mathsf{T}| \\ &\leq \sum_f \sqrt{w_f/w_e} \cdot \sqrt{w_e w_f} |b_e L^\dagger b_f^\mathsf{T}| \\ &\leq \sqrt{\sum_f w_f/w_e} \cdot \sqrt{\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2} \qquad \text{(by Cauchy-Schwarz)}\end{aligned}$$

We now bound each of the two terms separately. For the first term,

$$1/w_e \cdot \sum_f w_f = (p_e + 1/m)^{-1} \cdot \sum_f \left(p_f + 1/m\right) \leq 2 \cdot (p_e + 1/m)^{-1} \leq 2 \cdot 1/(1/m) = 2m.$$

For the second term, by Lemma 5.5.8, we know that

$$\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \leq 1.$$

Putting these two inequalities together, we get that $\text{stretch}_w(e) \leq \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. □

With these two lemmas, the rest of the proofs of the bound on the competitive ratio are exactly the same as in the ℓ_∞ case. While most of the running time analysis holds, note that we need to sketch slightly different matrices now. Earlier, we wanted to approximate $\text{load}_w(e) = w_e \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}|$. Thus we approximated $\|BL^\dagger b_e\|_1$ with our sketch matrix, and then multiplied it with w_e to obtain approximate loads. Since we now need to approximate $\text{stretch}_w(e) = \sum_f w_f |b_e L^\dagger b_f^\mathsf{T}|$, we instead sketch $WBL^\dagger b_e$.

The main thing is to note that load and stretch are simply the row and column 1-norms respectively of $WBL^\dagger B^\mathsf{T}$.

5.8. The Representation of Oblivious Routing and the Parallel Complexity

Representation of Oblivious Routing. Any *linear* oblivious routing scheme can be implemented using the following representation: every edge $e \in E$ stores the flow sent across edge e by an oblivious routing that sends one unit of flow from u to x, for every vertex $u \in V$ and some arbitrary but fixed target vertex $x \in V$. We let $f_{u,x}(e)$ denote the value of such an oblivious flow. Thus every edge stores n values and the total space requirement is $O(nm)$. Upon receiving a query for routing demand pairs $\{d_{s,t}\}$ of some demand vector $d \in \mathbb{R}^{\binom{n}{2}}$, we can compute the (s,t)-flow along any edge $e \in E$ by

$$d_{s,t} \cdot (f_{s,x}(e) - f_{t,x}(e)),$$

where correctness follows from the oblivious routing operator being linear.

We need to show that our oblivious routing operator based on a convex combination of electrical routings is linear. Note that $M_w := WBL^\dagger$ is a linear routing operator for any weighting on the edges $\{w_e\}$. Therefore, any convex combination $\sum_i \lambda_i M_{w_i}$ is also a linear routing operator.

This is the time spent constructing the required data structures before one can start routing packets using the oblivious routing.

It remains to study the running time of constructing such a representation, which we refer to as the *preprocessing time*. We start by considering the cost of constructing the representation for a single electrical routing $M_w := WBL^\dagger$. Since $f_{u,x}(e) = (WBL^\dagger \chi_{u,x})_e$, our goal is to compute $WBL^\dagger \chi_{u,x}$, for every $u \in V$ and the fixed vertex x, which can be achieved by solving n Laplacian systems. By Theorem 5.9.1, each such system can be solved in $\widetilde{O}(m)$ time, which in turn leads to a processing time of $\widetilde{O}(mn)$ for a single electrical routing. Since our oblivious routing scheme consists of $O(\sqrt{m})$ electrical routings, it follows that the total preprocessing time for constructing the representation for these electrical routings is $\widetilde{O}(m^{3/2}n)$.

Parallel Complexity. We next bound the parallel complexity of the multiplicative weights updates algorithm for computing the weights of our oblivious routing scheme and the algorithm for computing the representations of our scheme. Both of these rely on the fact that a Laplacian system can be solved to high accuracy with $\widetilde{O}(m)$ work and polylogarithmic depth.

[88]: Peng et al. (2014), "An efficient parallel solver for SDD linear systems"

[89]: Kyng et al. (2016), "Sparsified Cholesky and multigrid solvers for connection laplacians"

Theorem 5.8.1 ([88, 89]) *Given a n-vertex m-edge graph G, a Laplacian matrix L, a demand vector $y \in \mathbb{R}^n$ and an error bound ε_L, there is a parallel algorithm that achieves $\widetilde{O}(m)$ work and $\widetilde{O}(1)$ depth and returns a vector $x \in \mathbb{R}^n$ such that*

$$\|x - L^\dagger y\|_L \leq \varepsilon_L \cdot \|L^\dagger y\|_L.$$

Our first result shows that our MWU-based oblivious routing can be implemented in parallel with $\widetilde{O}(m^{3/2})$ work and $\widetilde{O}(\sqrt{m})$ depth.

Lemma 5.8.2 *There is a parallel implementation of Algorithm 5.1 that achieves $\widetilde{O}(m^{3/2})$ work and $\widetilde{O}(\sqrt{m})$ depth.*

Proof. We start by analyzing the parallel complexity of Algorithm 5.1. Consider one iteration of the **for** loop, and observe that the parallel complexity is dominated by the parallel cost of computing approximate loads in Line 9, i.e., Algorithm 5.2. Hence, it suffices to bound the parallel complexity of the latter. For generating the sketch matrix C, note that each edge $e \in E$ draws $\ell = O(\log n)$ independent random variables from the Cauchy distribution [82], and since these operations can be performed locally, it follows that constructing C takes $\widetilde{O}(m\ell) = \widetilde{O}(m)$ work and $O(1)$ depth.

[82]: Indyk (2006), "Stable distributions, pseudorandom generators, embeddings, and data stream computation"

Next, we compute the $(n \times \ell)$-dimensional matrix $X = B^\mathsf{T} C^\mathsf{T}$ in a row-wise fashion (Algorithm 5.2, Line 4). The u-th row of B^T contains $\deg(u)$ non-zero entries, and thus an entry $(X)_{u,i} = (B^\mathsf{T} C^\mathsf{T})_{u,i} = \sum_{e|e\sim u} b^\mathsf{T}_{u,e} c^\mathsf{T}_{e,i}$ can be evaluated with $O(\deg(u))$ work and $O(\log n)$ depth. As X has only $\ell = O(\log n)$ columns, it follows that u-th row can be computed with $\widetilde{O}(\deg(u))$ work and $O(\log n)$ depth. Summing the costs over all rows of X, we conclude that X can be computed with $\widetilde{O}\left(\sum_u \deg(u)\right) = \widetilde{O}(m)$ work and $O(\log n)$ depth. Finally, Lines 5 and 6 of Algorithm 5.2 involve solving $\ell = O(\log n)$ Laplacian systems. By Theorem 5.8.1, we can solve all these systems in parallel with $\widetilde{O}(m)$ work and $\widetilde{O}(1)$ depth.

Bringing all the above bounds together shows that an iteration of the **for** loop in Algorithm 5.1 can be implemented in parallel with $\widetilde{O}(m)$ work and $\widetilde{O}(1)$ depth. Since in total there are $\widetilde{O}(\sqrt{m})$ iterations, we get that a parallel implementation of Algorithm 5.1 has $\widetilde{O}(m^{3/2})$ work and $\widetilde{O}(\sqrt{m})$ depth. □

Our second result show that the representation of our oblivious routing can be implemented in parallel with $\widetilde{O}(m^{3/2}n)$ work and $\widetilde{O}(1)$ depth.

Lemma 5.8.3 *The representation of the oblivious routing based on $O(\sqrt{m})$ electrical routings can be implemented in parallel with $\widetilde{O}(m^{3/2}n)$ work and $\widetilde{O}(1)$ depth.*

Proof. We first analyze the cost of a single electrical routing given a weighting of the edges. Recall that our representation requires that every vertex solves one Laplacian system. These systems can be solved independently of each other (and thus in parallel), and by Theorem 5.8.1, each of them can be implemented in parallel with $\widetilde{O}(m)$ work and $\widetilde{O}(1)$ depth. Thus, the parallel complexity of a single electrical routing is $\widetilde{O}(mn)$ work and $\widetilde{O}(1)$ depth.

Now, note that our MWU algorithm has already computed $\widetilde{O}(\sqrt{m})$ weightings of the graph, each corresponding to a single electrical routing. Once computed, these weightings are independent of each other and, thus, the electrical routings (one per vertex) for all of these weightings can be computed in parallel. Therefore, the total complexity for computing the representation of our routing scheme is $\widetilde{O}(m^{3/2}n)$ work and $\widetilde{O}(1)$ depth.

To evaluate the average (oblivious) flow from the convex combination of $\widetilde{O}(\sqrt{m})$ electrical routings, each edge can locally compute the convex combination of the oblivious flows sent along that edge with $\widetilde{O}(\sqrt{m})$ work and $O(\log(\sqrt{m})) = \widetilde{O}(1)$ depth. Thus, it follows that the total parallel complexity of this step is $\widetilde{O}(m^{3/2})$ work and $\widetilde{O}(1)$ depth.

Bringing the above bounds together proves the lemma. □

5.9. Algorithm using approximate Laplacian solvers

We give in Algorithm 5.3 a version of GetApproxLoad that uses an approximate Laplacian solver [90] instead of an exact one. The change is in Line 6. For concreteness, we use the following state-of-the-art approximate Laplacian solver by Jambulapati and Sidford [91].

[90]: Spielman et al. (2004), "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems"

[91]: Jambulapati et al. (2021), "Ultrasparse Ultrasparsifiers and Faster Laplacian System Solvers"

Theorem 5.9.1 ([91]) *There is an algorithm ApproxLapSolve that gets as input a Laplacian L of an n-vertex m-edge graph, a vector $y \in \mathbb{R}^n$, error bound ε_L, and returns a vector $x \in \mathbb{R}^n$ such that the following holds with probability $\geq 1 - 1/\mathrm{poly}(n)$.*

$$\|x - L^\dagger y\|_L \leq \varepsilon_L \cdot \|L^\dagger y\|_L$$

where $\|x\|_L = \sqrt{x^\mathsf{T} L x}$ is the norm induced by the Laplacian. Further, the algorithm runs in time $\widetilde{O}(m \log(1/\varepsilon_L))$.

Lemma 5.9.2 *For any approximation factor $0 < \varepsilon < 1$, and any weighted graph (G, w), let $\mathrm{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\mathrm{aload}_w = \textsc{GetApproxLoad}(G, w, \varepsilon)$ be the approximate loads computed by the algorithm when using an approximate Laplacian solver. Then with probability $\geq 1 - 1/\mathrm{poly}(n)$,*

$$(1 - \varepsilon) \cdot \mathrm{load}_w(e) \leq \mathrm{aload}_w(e) \leq (1 + \varepsilon) \cdot \mathrm{load}_w(e) \qquad \text{for all } e \in E$$

We prove this lemma in Section 5.9.1, and then explain the changes to the algorithm in Section 5.9.2.

Algorithm 5.3: GetApproxLoad, to compute approximate loads for electrical routing.

Input: A graph G, weights $\{w_e\}_{e\in E}$ on the edges, approximation factor ε.
Output: Approximation $\{\text{aload}_w\}_{e\in E}$ to the load on the edges.

Let B be the incidence edge-vertex matrix of G
Let $L := B^\intercal \operatorname{diag}(w) B$ be the Laplacian matrix
Set $C \leftarrow \text{SketchMatrix}(m, n^{-10}, \varepsilon/2)$
Set $X \leftarrow B^\intercal C^\intercal$
Let $X^{(i)}$ be the i^{th} column of X for all $i \in [\ell]$
Set $\widetilde{U}^{(i)} \leftarrow \text{ApproxLapSolve}(L, X^{(i)}, n^{-10}, \varepsilon_L)$ for all $i \in [\ell]$
Set $\widetilde{U} \leftarrow (\widetilde{U}^{(1)}, \widetilde{U}^{(2)}, \ldots, \widetilde{U}^{(\ell)})$ ▷ $\widetilde{U} \approx (CBL^\dagger)^\intercal$
Set $\text{aload}_w(e) \leftarrow w_e \cdot \text{RecoverNorm}(\widetilde{U}^\intercal b_e)$ for all $e \in E$
return aload_w

5.9.1. Proof of Lemma 5.9.2

We assume without loss of generality that $L^\dagger x \perp \mathbb{1}$ and $y \perp \mathbb{1}$, where $\mathbb{1}$ is the the all-ones vector, x is the input to ApproxLapSolve and y is the output received. This is because shifting a vector of potentials by a constant vector does not affect the guarantees in the L norm.

We will use the following two properties of the functions SketchMatrix and RecoverNorm in our analysis:

- $\text{RecoverNorm}(x_1, x_2, \ldots, x_\ell) = \text{median}(|x_1|, |x_2|, \ldots, |x_\ell|)$.
- $\max_{i,j} C_{ij} \le \text{poly}(m)$.

We prove Lemma 5.9.2 assuming the following lemmas.

This lemma converts the guarantee we have on the L norm to a guarantee on the ℓ_∞ norm.

Lemma 5.9.3 *Suppose we have two vectors $x, y \in \mathbb{R}^n$ such that $x, y \perp \mathbb{1}$ and $\|x - y\|_L \le \varepsilon \|y\|_L$. Then $\|x - y\|_\infty \le \varepsilon \cdot 2n^3 \cdot \|y\|_\infty$.*

The previous lemma gives us an ℓ_∞ guarantee for the columns of our matrix $\widetilde{U}$, and we convert this to a guarantee on the rows of $\widetilde{U}$ using this lemma.

Lemma 5.9.4 *Let U be an $n \times \ell$ matrix, with K being the largest value present in the matrix. Let $\widetilde{U}$ be an approximation of U such that for every $j \in [\ell]$, the j-th column $\widetilde{C}_j$ of $\widetilde{U}$ is ε-close to the j-th column C_j of U in the following sense: $\forall j \in [\ell], \|\widetilde{C}_j - C_j\|_\infty \le \varepsilon \cdot \|C_j\|_\infty$. Then for any $i \in [n]$, and rows R_i of U and $\widetilde{R}_i$ of $\widetilde{U}$,*

$$\|\widetilde{R}_i - R_i\|_\infty \le \varepsilon \cdot K$$

This lemma shows that RecoverNorm works well on approximate sketched vectors.

Lemma 5.9.5 *Let $a \in \mathbb{R}$ be a real number. Suppose $x \in \mathbb{R}^\ell$ satisfies*

$$(1 - \varepsilon) \cdot a \le \text{median}(|x_1|, |x_2|, \ldots, |x_\ell|) \le (1 + \varepsilon) \cdot a$$

Suppose $y \in \mathbb{R}^\ell$ is such that $\|x - y\|_\infty \le \varepsilon'$, then y satisfies

$$(1 - \varepsilon) \cdot a - \varepsilon' \le \text{median}(|y_1|, |y_2|, \ldots, |y_\ell|) \le (1 + \varepsilon) \cdot a + \varepsilon'$$

Our final lemma gives a lower bound on the effective resistance to convert an additive approximation guarantee into a multiplicative one. This follows from Rayleigh's monotonicity law and is shown, for example, by Spielman and Srivastava [52, Proposition 10].

Lemma 5.9.6 *For any weighted graph (G, w), the load on any edge e has the following lower bound.*

$$\text{load}_w(e) \ge \frac{2w_e}{nw_{\max}},$$

where $w_{\max}$ is the maximum edge weight.

[52]: Spielman et al. (2011), "Graph Sparsification by Effective Resistances"

We first use these four lemmas to prove Lemma 5.9.2.

Lemma 5.9.2 *For any approximation factor $0 < \varepsilon < 1$, and any weighted graph (G, w), let $\text{load}_w = \left(w_e \sum_f |b_e L^\dagger b_f^\mathsf{T}|\right)_{e \in E}$ be the true loads, and $\text{aload}_w = \textsc{GetApproxLoad}(G, w, \varepsilon)$ be the approximate loads computed by the algorithm when using an approximate Laplacian solver. Then with probability $\geq 1 - 1/\text{poly}(n)$,*

$$(1-\varepsilon) \cdot \text{load}_w(e) \leq \text{aload}_w(e) \leq (1+\varepsilon) \cdot \text{load}_w(e) \qquad \text{for all } e \in E$$

Proof. Let $U = L^\dagger B^\mathsf{T} C^\mathsf{T}$, and $\widetilde{U} = \textsc{ApproxLapSolve}(L, B^\mathsf{T} C^\mathsf{T}, \delta, \varepsilon_L)$ where we abuse notation to mean that ApproxLapSolve runs on each column of $B^\mathsf{T} C^\mathsf{T}$ and returns a column vector. Let R_i and $\widetilde{R}_i$ denote the rows, and C_j and $\widetilde{C}_j$ denote the columns of U and $\widetilde{U}$ respectively. We have

$$\begin{aligned}
&\|\widetilde{C}_j - C_j\|_L \leq \varepsilon_L \cdot \|C_j\|_L, \text{ and therefore} && \text{(by guarantees of Laplacian solver)}\\
&\|\widetilde{C}_j - C_j\|_\infty \leq \varepsilon_L \cdot 2n^3 \cdot \|C_j\|_\infty && \text{(by Lemma 5.9.3)}
\end{aligned}$$

With K as an upper bound on the maximum entry in $CBL^\dagger$, and using the above inequality in Lemma 5.9.4, we get that

$$\|\widetilde{R}_i - R_i\|_\infty \leq \varepsilon_L \cdot 2n^3 \cdot K \tag{5.2}$$

Note that for any $e = (u, v)$, $(U^\mathsf{T} b_e)^\mathsf{T} = R_u - R_v$, and $(\widetilde{U}^\mathsf{T} b_e)^\mathsf{T} = \widetilde{R}_u - \widetilde{R}_v$. Thus

$$\begin{aligned}
\|(U^\mathsf{T} b_e) - (\widetilde{U}^\mathsf{T} b_e)\|_\infty &= \|(R_u - R_v) - (\widetilde{R}_u - \widetilde{R}_v)\|_\infty \\
&\leq \|R_u - \widetilde{R}_u\|_\infty + \|R_v - \widetilde{R}_v\|_\infty \\
&\leq 4\varepsilon n^3 K && \text{(by Equation 5.2)} \\
&\leq \frac{\varepsilon}{2mn}. && \text{(by choice of } \varepsilon_L = \varepsilon/(8mn^4K))
\end{aligned}$$

Since the guarantees of RecoverNorm gives us

$$\left(1 - \frac{\varepsilon}{2}\right) \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| \leq \textsc{RecoverNorm}(U^\mathsf{T} b_e) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}|,$$

using Lemma 5.9.5 on $U^\mathsf{T} b_e$ and $\widetilde{U}^\mathsf{T} b_e$, we get

$$\left(1 - \frac{\varepsilon}{2}\right) \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| - \frac{\varepsilon}{2mn} \leq \textsc{RecoverNorm}(\widetilde{U}^\mathsf{T} b_e) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \sum_f |b_e L^\dagger b_f^\mathsf{T}| + \frac{\varepsilon}{2mn}.$$

Thus,

$$\left(1 - \frac{\varepsilon}{2}\right) \cdot \text{load}_w(e) - \frac{\varepsilon w_e}{2mn} \leq \textsc{GetApproxLoad}(G, w) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \text{load}_w(e) + \frac{\varepsilon w_e}{2mn}$$

Since for any edge $f \in E$, we have $w_f = \left(p_f + 1/m\right)^{-1} \leq m$, we get $w_{\max} \leq m$. Lemma 5.9.6 gives us that $\text{load}_w(e) \geq w_e/mn$, which proves the lemma. □

We now prove the intermediate lemmas.

Lemma 5.9.7 *For any $x \in \mathbb{R}^n$, $\|x\|_L \leq 2n \cdot \|x\|_\infty$.*

Proof. Since $\|x\|_\infty^2 \le 1$ implies $|x_i| \le 1$ for all $i \in [n]$, and since $\|x\|_L^2 = x^\mathsf{T} L x$, we have

$$\max_{x \in \mathbb{R}^n} \frac{\|x\|_L^2}{\|x\|_\infty^2} \le \max_{x \in \mathbb{R}^n : |x_i| \le 1} x^\mathsf{T} L x$$

Since $x^\mathsf{T} L x = \sum_{(u,v)\in E} (x_u - x_v)^2$, and $|x_i| \le 1$, each term is bounded by 4. Thus, the sum is at most $4m \le 4n^2$, giving the lemma. □

Lemma 5.9.8 *For any $x \in \mathbb{R}^n$ such that $x \perp \mathbb{1}$, we have $n^{-2} \cdot \|x\|_\infty \le \|x\|_L$.*

Proof. Since $\min_{x \in \mathbb{R}^n} \|x\|_L^2 / \|x\|_2^2 = 0$ which is attained by $\mathbb{1}$, by the variational characterization of eigenvalues of a symmetric matrix, the second smallest eigenvalue λ_2 of L is given by

$$\min_{x \in \mathbb{R}^n : x \perp \mathbb{1}} \frac{\|x\|_L^2}{\|x\|_2^2} = \lambda_2$$

By Cheeger's inequality, $\lambda_2 \ge h^2/2\Delta$, where $h = \min_{S : |S| \le n/2} |E(S, \bar{S})|/|S|$ is the unnormalized Cheeger's constant and Δ is the maximum degree in the graph. Thus

$$\lambda_2 \ge h^2 \cdot \frac{1}{2\Delta} \ge \frac{4}{n^2} \cdot \frac{1}{2n} \ge n^{-3}$$

Thus for any $x \in \mathbb{R}^n$ such that $x \perp \mathbb{1}$, $\|x\|_L \ge n^{-3/2} \cdot \|x\|_2 \ge n^{-3/2} \cdot \|x\|_\infty$, and the lemma follows. □

Lemma 5.9.3 *Suppose we have two vectors $x, y \in \mathbb{R}^n$ such that $x, y \perp \mathbb{1}$ and $\|x - y\|_L \le \varepsilon \|y\|_L$. Then $\|x - y\|_\infty \le \varepsilon \cdot 2n^3 \cdot \|y\|_\infty$.*

Proof. By Lemma 5.9.7 and Lemma 5.9.8, we have

$$n^{-2} \cdot \|x - y\|_\infty \le \|x - y\|_L \qquad \text{and} \qquad \|y\|_L \le 2n \cdot \|y\|_\infty$$

Together with the assumption in our lemma, we get

$$\|x - y\|_\infty \le n^2 \cdot \|x - y\|_L \le \varepsilon \cdot n^2 \cdot \|y\|_L \le \varepsilon \cdot 2n^3 \cdot \|y\|_\infty.$$ □

Lemma 5.9.4 *Let U be an $n \times \ell$ matrix, with K being the largest value present in the matrix. Let $\widetilde{U}$ be an approximation of U such that for every $j \in [\ell]$, the j-th column $\widetilde{C}_j$ of $\widetilde{U}$ is ε-close to the j-th column C_j of U in the following sense: $\forall j \in [\ell], \|\widetilde{C}_j - C_j\|_\infty \le \varepsilon \cdot \|C_j\|_\infty$. Then for any $i \in [n]$, and rows R_i of U and $\widetilde{R}_i$ of $\widetilde{U}$,*

$$\|\widetilde{R}_i - R_i\|_\infty \le \varepsilon \cdot K$$

Proof. For any $i \in [n]$,

$$\begin{aligned}
\|\widetilde{R}_i - R_i\|_\infty &\le \max_{i \in [n]} \|\widetilde{R}_i - R_i\|_\infty \\
&= \max_{i,j \in [n] \times [\ell]} |\widetilde{U}_{ij} - U_{ij}| \\
&= \max_{j \in [\ell]} \|\widetilde{C}_j - C_j\|_\infty \\
&\le \varepsilon \cdot \max_{j \in [\ell]} \|C_j\|_\infty \qquad \text{(by assumption on } \{C_j\} \text{ and } \{\widetilde{C}_j\}) \\
&\le \varepsilon \cdot K.
\end{aligned}$$ □

Lemma 5.9.9 *Let $x, y \in \mathbb{R}^\ell$ be such that $\|x - y\|_\infty \leq \varepsilon$, where ℓ is odd. Then*

$$|\textsc{median}(|y_1|, |y_2|, \ldots, |y_\ell|) - \textsc{median}(|x_1|, |x_2|, \ldots, |x_\ell|)| \leq \varepsilon$$

Proof. Note that if $|x_i - y_i| \leq \varepsilon$, then $||x_i| - |y_i|| \leq \varepsilon$ as well. Thus we can assume without loss of generality that x and y are non-negative.

Relabel the indices such that $x_1 \leq x_2 \leq \ldots \leq x_\ell$ are in non-decreasing order. The assumption on ℓ_∞ norm gives us that for all $i \in [\ell]$,

$$|x_i - y_i| \leq \varepsilon \tag{5.3}$$

Let π be a permutation such that $y_{\pi(1)} \leq y_{\pi(2)} \leq \ldots \leq y_{\pi(\ell)}$ is in non-decreasing order. Let $m = \lceil \ell/2 \rceil$. We want to show that $|y_{\pi(m)} - x_m| \leq \varepsilon$. If $m = \pi(m)$, then the lemma follows from Equation (5.3) with $i = m$. So assume that $m \neq \pi(m)$.

Consider the case when $m > \pi(m)$. Then since $y_m \geq y_{\pi(m)}$, we have $x_m \geq y_m - \varepsilon \geq y_{\pi(m)} - \varepsilon$. For the other direction, since $m > \pi(m)$, there exists an index j such that $x_j > x_m$ and $y_j \leq y_{\pi(m)}$. For this index, we thus get $y_{\pi(m)} \geq y_j \geq x_j - \varepsilon \geq x_m - \varepsilon$. The case when $m < \pi(m)$ is symmetric. □

Lemma 5.9.5 *Let $a \in \mathbb{R}$ be a real number. Suppose $x \in \mathbb{R}^\ell$ satisfies*

$$(1 - \varepsilon) \cdot a \leq \textsc{median}(|x_1|, |x_2|, \ldots, |x_\ell|) \leq (1 + \varepsilon) \cdot a$$

Suppose $y \in \mathbb{R}^\ell$ is such that $\|x - y\|_\infty \leq \varepsilon'$, then y satisfies

$$(1 - \varepsilon) \cdot a - \varepsilon' \leq \textsc{median}(|y_1|, |y_2|, \ldots, |y_\ell|) \leq (1 + \varepsilon) \cdot a + \varepsilon'$$

Proof. We have by the assumption on x that

$$\begin{aligned}
(1 - \varepsilon) \cdot a - \varepsilon' &\leq \textsc{median}(|x_1|, |x_2|, \ldots, |x_\ell|) - \varepsilon' \\
&\leq \textsc{median}(|y_1|, |y_2|, \ldots, |y_\ell|) && \text{(by Lemma 5.9.9)} \\
&\leq \textsc{median}(|x_1|, |x_2|, \ldots, |x_\ell|) + \varepsilon' && \text{(by Lemma 5.9.9)} \\
&\leq (1 + \varepsilon) \cdot a + \varepsilon'
\end{aligned}$$

□

5.9.2. Changes to the algorithm

The algorithm is very similar to the case with an exact Laplacian solver, with minor changes. We detail them for completeness. Since the maximum entry in C is upper bounded by $\text{poly}(n, m)$, the maximum entry in $CBL^\dagger$ is upper bounded by some $\text{poly}(n, m)$ as well. We call this value as $K = \max_{i,j}(CBL^\dagger)_{ij}$. We use $\varepsilon_L = \varepsilon/(8mn^4K)$.

The only change in the running time from the earlier algorithm is the time taken by the approximate Laplacian solver. Recall that the running time of the approximate Laplacian solver is $\widetilde{O}(m \log(1/\varepsilon_L))$. Since the maximum entry in C is bounded by $\text{poly}(m)$, the maximum entry in $CBL^\dagger$ is bounded by $\text{poly}(n, m)$ as well. We denote this upper bound as K. Thus $\varepsilon_L \geq 1/\text{poly}(n)$, and the approximate Laplacian solver runs in time $\widetilde{O}(m)$.

5.10. Routing on capacitated graphs

Capacitated Graph. A capacitated graph $G = (V, E, u)$ is an undirected graph with a function $u : E \to \mathbb{R}^+$ that represents the edge capacities.

Congestion. Given a flow $f \in \mathbb{R}^m$, the congestion of an edge is the amount of flow on that edge relative to its capacity, given by $|f_e|/u_e$.

Let U denote the $m \times m$ diagonal matrix with u_e on the diagonals. Kelner and Maymounkov [65, Theorem 3.1] show that the worst-case demands for a capacitated graph is u_e along each edge, i.e., the columns of $B^\mathsf{T}U$. Note that these can be routed optimally with congestion 1, by simply routing each demand of u_e across the same edge. In their presentation of the proof, they use $\{w_e\}$ for both the capacities and the conductances. We, on the other hand, need to use conductances that are different from the capacities. While their proof works for our case, for the sake of clarity, we present their proof of worst-case demands in Section 5.10.4 using our notation.

Load. For a linear oblivious routing M, the congestion of edge e for routing $u_f \cdot b_f^\mathsf{T}$ is given by $\text{load}_M(f \to e) = u_e^{-1} \cdot u_f \cdot |(Mb_f^\mathsf{T})_e|$. The load on edge e is then given by summing this congestion up for each $f \in E$, giving

While $\text{cong}_M(e)$ would be better than $\text{load}_M(e)$ in the capacitated case, we continue using load_M for continuity with the main body.

$$\text{load}_M(e) = \sum_f \text{load}_M(f \to e) = u_e^{-1} \cdot \sum_f u_f \cdot |(Mb_f^\mathsf{T})_e|$$

For an electrical flow with conductances $\{w_e\}$, its oblivious routing is $WBL^\dagger$ (the Laplacian $L = B^\mathsf{T}WB$ is with respect to W), and the load is

$$\text{load}_w(e) = \frac{w_e}{u_e} \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}|$$

To show the competitive ratio bound for capacitated graphs, we then need to give a set of weights $\{w_e\}$ for each $p_e \in \Delta^m$ such that the MWU algorithm can be performed, and we need to show that we can still use sketching to get approximate loads in $\widetilde{O}(m)$ time.

Concretely, we will use the weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ in the algorithm.

We need to prove analogues of Lemma 5.5.1, Lemma 5.5.2, and we need to provide a version of GetApproxLoad that works in the capacitated case. We do so in the next three sections. This, then, will give us Theorem 5.1.1.

5.10.1. Bound on Average Loads

Lemma 5.10.1 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ satisfies $\sum_e p_e \text{load}_w(e) \leq 2\alpha_{LOCAL}$.*

Proof. Setting ℓ_e to $u_e/\sqrt{w_e}$ and applying Lemma 5.2.1, we get

$$\begin{aligned}
\sum_e p_e \mathrm{load}_w(e) &= \sum_e p_e \cdot \sum_f \mathrm{load}_w(f \to e) \\
&= \sum_e p_e \cdot {}^{w_e}/_{u_e} \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of load)} \\
&= \sum_e p_e \cdot u_e \cdot (p_e + 1/m)^{-1} \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } w_e\text{)} \\
&\le \sum_e \sum_f u_e u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \sum_{e,f} {}^{u_e u_f}/_{\sqrt{w_e w_f}} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&= \sum_{e,f} \ell_e \ell_f \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by choice of } \ell_e\text{)} \\
&\le \alpha_{\mathrm{LOCAL}} \cdot \|\ell\|_2^2 && \text{(by Lemma 5.2.1)} \\
&= \alpha_{\mathrm{LOCAL}} \cdot \sum_e {}^{u_e^2}/_{w_e} \\
&= \alpha_{\mathrm{LOCAL}} \cdot \sum_e (p_e + 1/m) \\
&= 2\alpha_{\mathrm{LOCAL}}. && \square
\end{aligned}$$

5.10.2. Bound on Width

Lemma 5.10.2 *For any probability distribution $p \in \Delta^m$, the oblivious routing M_w corresponding to the electrical network with weights $w_e = u_e^2 \cdot (p_e + 1/m)^{-1}$ satisfies $\mathrm{load}_w(e) \le \sqrt{2m}$ for every edge e.*

Proof. By definition of w_e, we have $u_e = \sqrt{w_e \cdot (p_e + 1/m)}$. Fixing an edge e,

$$\begin{aligned}
\mathrm{load}_w(e) &= {}^{w_e}/_{u_e} \cdot \sum_f u_f \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&\le \sqrt{{}^{w_e}/_{(p_e+1/m)}} \cdot \sum_f \sqrt{w_f \cdot \left(p_f + 1/m\right)} \cdot |b_e L^\dagger b_f^\mathsf{T}| && \text{(by definition of } u_e \text{ and } u_f\text{)} \\
&\le \sum_f \sqrt{{}^{(p_f+1/m)}/_{(p_e+1/m)}} \cdot \sqrt{w_e w_f} \cdot |b_e L^\dagger b_f^\mathsf{T}| \\
&\le \sqrt{\sum_f {}^{(p_f+1/m)}/_{(p_e+1/m)}} \cdot \sqrt{\sum_f w_e w_f \cdot |b_e L^\dagger b_f^\mathsf{T}|^2} && \text{(by Cauchy-Schwarz)}
\end{aligned}$$

We now bound each of the two terms separately. For the first term, note that

$$(p_e + 1/m)^{-1} \cdot \sum_f \left(p_f + 1/m\right) \le 2 \cdot (p_e + 1/m)^{-1} \le 2 \cdot {}^1/_{(1/m)} = 2m.$$

For the second term, by Lemma 5.5.8, we know that

$$\sum_f w_e w_f |b_e L^\dagger b_f^\mathsf{T}|^2 \le 1.$$

Putting these two inequalities together, we get that $\mathrm{load}_w(e) \le \sqrt{2m}$ for any edge e, which gives the desired bound of $\sqrt{2m}$ on the width. $\square$

5.10.3. Capacitated GetApproxLoad

Algorithm 5.4 contains the changes to GetApproxLoad for the capacitated case. Correctness follows from noting that $\text{load}_w(e)$ is the ℓ_1 norm of the e^{th} row of $U^{-1}WBL^{\dagger}B^{\mathsf{T}}U$.

Algorithm 5.4: GetApproxLoad, to compute approximate loads for electrical routing.

Input: A graph G, weights $\{w_e\}_{e\in E}$ and capacities $\{u_e\}_{e\in E}$ on the edges, approximation factor ε.
Output: Approximation $\{\text{aload}_w(e)\}_{e\in E}$ to the load on the edges.

Let B be the edge-vertex incidence matrix of G
Let $L := B^{\mathsf{T}}\,\text{diag}(w)B$ be the Laplacian matrix
Set $C \leftarrow \text{SketchMatrix}(m, n^{-10}, \varepsilon)$
Set $X \leftarrow B^{\mathsf{T}}UC^{\mathsf{T}}$
Let $X^{(i)}$ be the i^{th} column of X for all $i \in [\ell]$
Set $V^{(i)} \leftarrow \text{LapSolve}(L, X^{(i)})$ for all $i \in [\ell]$
Set $V \leftarrow (V^{(1)}, V^{(2)}, \ldots, V^{(\ell)})$ ▷ $V = (CUBL^{\dagger})^{\mathsf{T}}$
Set $\text{aload}_w(e) \leftarrow w_e u_e^{-1} \cdot \text{RecoverNorm}(U^{\mathsf{T}}b_e)$ for all $e \in E$
return aload_w

5.10.4. Worst-case demands

We present the proof of the worst-case demands for oblivious routing on a capacitated graph $G = (V, E, u)$ being $B^{\mathsf{T}}U$ from Kelner and Maymounkov [65, Theorem 3.1], using $\{u_e\}$ as capacities, and M as a linear oblivious routing.

Since congestion of a linear oblivious routing scales linearly when the demands are multiplicatively increased, it suffices to consider demands that can be routed optimally with congestion 1. Let $\{\chi_i\}_i$ be such a set of demands, and let $\{f_i\}_i$ be its optimal routing. The claim follows from noting that demands $\{\sum_i |f_{i,e}|\}_e$, i.e., demands of $\sum_i |f_{i,e}|$ across each edge e, can still be (non-linearly) routed with congestion 1.

for any $\{\chi_i\}_i$ with $\text{OPT}(\{\chi_i\}_i) = 1$

$$\beta_\infty(M) \le \max_{e'} u_{e'}^{-1} \cdot \sum_i |b_{e'} M \chi_i|$$

(since f routes $\{\chi_i\}_i$)

$$= \max_{e'} u_{e'}^{-1} \cdot \sum_i \left| b_{e'} M \left(\sum_e f_{i,e} b_e^{\mathsf{T}} \right) \right|$$

(by linearity of M)

$$= \max_{e'} u_{e'}^{-1} \cdot \sum_i \left| \sum_e f_{i,e} \cdot b_{e'} M b_e^{\mathsf{T}} \right|$$

(since $|\sum \cdot| \le \sum |\cdot|$)

$$\le \max_{e'} u_{e'}^{-1} \cdot \sum_{i,e} \left| f_{i,e} \cdot b_{e'} M b_e^{\mathsf{T}} \right|$$

$$= \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| \sum_i |f_{i,e}| \cdot b_{e'} M b_e^{\mathsf{T}} \right|$$

(since $\{f_i\}_i$ has congestion 1)

$$\le \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| u_e \cdot b_{e'} M b_e^{\mathsf{T}} \right|$$

$$= \max_{e'} u_{e'}^{-1} \cdot \sum_e \left| \cdot b_{e'} M (u_e b_e^{\mathsf{T}}) \right|$$

$$= \|U^{-1} M B^{\mathsf{T}} U\|_\infty.$$

6. Lower Bounds for Dynamic Graphs

You can only do what you can, no matter how you try.

Kōbō Abe, *The Box Man*

6.1. Introduction

The introduction of strong conditional lower bounds based on widely-believed complexity assumptions [92, 93] has had a fundamental influence on the field of dynamic graph algorithms, marking the boundary between tractable and intractable problems. However, graphs arising in real-world applications often differ significantly from the very specifically crafted graphs for which the lower bound results are shown. Frequently, real-world graphs have some special structure, such as having a power-law degree distribution, a constant degree, or being planar. Expanders, on the other side, have recently been used to design dynamic algorithms for *general* graphs. This naturally leads to the question of determining the complexity of dynamic graph algorithms for these graph classes, and this is exactly the question investigated in this chapter.

While the complexity of dynamic graph algorithms for planar graphs has already been studied quite extensively [92, 94–101], the question is still widely open for other families of graphs, including power-law graphs, constant-degree graphs, and expanders. Certain problems become easier for these graph classes: As an N-node[1] constant-degree graph has $O(N)$ edges, computing all-pairs shortest paths (APSP) takes only time $\tilde{O}(N^2)$, while the popular APSP conjecture postulates that any algorithm requires $N^{3-o(1)}$ expected time to compute APSP on general graphs in the word RAM model with $O(\log N)$-bit words. Moreover, some problems become trivial in these graph classes, e.g., computing shortest paths with logarithmic additive error on expander graphs is trivial, due to their low diameter.

In this chapter, we focus on graph problems that have real-world applications such as shortest-paths (which has applications in online navigation), matching (which has applications in reconfigurable datacenters), and densest subgraphs (which has applications in network analysis), yet we believe that our general approach can be applied to further graph problems. For these three problems, the known conditional lower bounds construct graphs that are far from being in the classes we consider: They have maximum degree $\Omega(N)$ and small cuts, and their degree distribution is unstructured as it depends on the hard instance one reduces from.

Constant-Degree Graphs. Various dynamic graph problems that admit strong lower bounds in general graphs have very efficient algorithms on constant-degree graphs. Let Δ be the maximum degree in the graph. For *local* problems, where the solution at a node v can be computed by simply analyzing information stored at the neighbors of v such as maintaining a maximal matching, a maximal independent set, or a $(\Delta+1)$-vertex coloring, there exist simple dynamic algorithms with $O(\Delta)$ update time and constant query time. Additionally, for various problems that count or detect certain fixed subgraphs with c nodes (such as triangle counting for $c = 3$) there

[92]: Abboud et al. (2016), "Popular Conjectures as a Barrier for Dynamic Planar Graph Algorithms"

[93]: Henzinger et al. (2015), "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture"

[94]: Subramanian (1993), "A Fully Dynamic Data Structure for Reachability in Planar Digraphs"

[95]: Henzinger et al. (1997), "Faster Shortest-Path Algorithms for Planar Graphs"

[96]: Klein et al. (1998), "A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs"

1: To avoid confusion with the parameter n and matrix M used in the online-matrix-vector multiplication conjecture, we use N to denote the number of vertices and m the number of edges in the dynamic graphs.

[97]: Italiano et al. (2011), "Improved algorithms for min cut and max flow in undirected planar graphs"

[98]: Abraham et al. (2012), "Fully Dynamic Approximate Distance Oracles for Planar Graphs via Forbidden-Set Distance Labels"

[99]: Abraham et al. (2016), "On Dynamic Approximate Shortest Paths for Planar Graphs with Worst-Case Costs"

[100]: Italiano et al. (2017), "Decremental single-source reachability in planar digraphs"

[101]: Charalampopoulos et al. (2022), "Single-source shortest paths and strong connectivity in dynamic planar graphs"

Table 6.1.: Counting problems which admit polynomial conditional lower bounds on general graphs (amortized) and on Erdős–Rényi graphs (average case), but have algorithms with constant update and query times in constant-degree graphs. For the lower bounds above, there is no dynamic algorithm with pre-processing time pre, update time upd, and query time qry unless the OMv conjecture is false. When pre is unspecified, poly(N) pre-processing time is allowed.

Problems	Lower bounds					Upper bounds	
	General graphs [93]		Erdős–Rényi avg-case [103]			constant Δ (trivial)	
	upd	**qry**	**pre**	**upd**	**qry**	**upd**	**qry**
Δ-count	$m^{1/2-\varepsilon}$	$m^{1-\varepsilon}$	$N^{3-\varepsilon}$	$m^{1/2-\varepsilon}$	$m^{1-\varepsilon}$	Δ	1
C_4-count						Δ^3	1
5-length (s,t)-path count			$N^{2-\varepsilon}$	$N^{\omega-2-\varepsilon}$	1	Δ^4	1

exists dynamic algorithms with $O(\Delta^{c-1})$ update time and constant query time, even though they have polynomial conditional lower bounds in general graphs (see Table 6.1). These efficient algorithms for local problems rule out the possibility of any non-trivial constant-degree lower bounds.

[102]: Gupta et al. (2013), "Fully Dynamic (1+ϵ)-Approximate Matchings"

Furthermore, even for the non-local problem of maintaining a maximum matching, Gupta and Peng [102] designed a $(1+\delta)$-approximation algorithm for any small $\delta > 0$ that runs in $O\left(\min\left\{\frac{m}{|M(t)|}, |M(t)|\right\}\delta^{-2}\right)$ amortized time per update, where $M(t)$ denotes the maximum cardinality matching after the t-th update operation. As in a graph with maximum degree Δ it holds that $|M(t)| \geq m/(2\Delta)$, their algorithm achieves an amortized update time of $O(\Delta\,\delta^{-2})$, which is $O(\delta^{-2})$ in constant-degree graphs. This raises the question of how efficiently other non-local dynamic graph problems such exact maximum matching, shortest paths, and densest subgraph can be solved in dynamic constant-degree graphs.

[103]: Henzinger et al. (2022), "The Complexity of Average-Case Dynamic Subgraph Counting"

Expanders. Expander decompositions are increasingly becoming a central tool for designing dynamic graph algorithms with improved running time bounds for various graph problems such as connectivity, minimum spanning tree, shortest paths, conductance, edge-connectivity, maximum flows, and minimum cuts [104–107]. One of the central subproblems that these algorithms have to handle is to solve a graph problem on a dynamically changing expander. To understand the limitations of this approach it is crucial to understand which problems can be solved efficiently on expanders, and which cannot. We present novel lower bounds for dynamic problems on expanders, more specifically on constant-degree expanders.

[104]: Jin et al. (2020), "Fully Dynamic c-Edge Connectivity in Subpolynomial Time"

[105]: Goranci et al. (2021), "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms"

[106]: Chuzhoy et al. (2021), "Deterministic Algorithms for Decremental Shortest Paths via Layered Core Decomposition"

[107]: Chuzhoy (2021), "Decremental all-pairs shortest paths in deterministic near-linear time"

These results also have an interesting connection to the average-case hardness of dynamic graph algorithms. Recently, lower bounds on the average-case hardness were shown for various subgraph counting problems in dynamic Erdős-Rényi graphs (see Table 6.1 for some of them) [103]. As random graphs are usually expanders, giving lower bounds for a problem on dynamic expanders gives an indication that this problem might also be hard in the average case and can motivate further work in this direction.

Power-law Graphs. In *power-law graphs*, the fraction of nodes with degree d is proportional to $1/d^c$ for some constant $c > 0$. Static and dynamic power-law graphs arise surprisingly often in real-world networks, such as the Internet, the world-wide web, and citation graphs, as well as in physics, linguistics, and economics. Even though the existence of large dynamic power-law graphs was already pointed out in 2004 [108], no efficient dynamic algorithms have been developed specifically for this class of graphs. This leads to the question of whether sub-polynomial time dynamic algorithms are even possible for power-law graphs. In fact, dynamic power-law

[108]: Graham (2004), "Large Dynamic Graphs: What Can Researchers Learn from Them?"

Problem	Class	Section	upd	qry
Maximum Matching	$\Delta \leq 3$	6.4.1	$m^{1/2-\varepsilon}$	$m^{1-\varepsilon}$
	constant Δ & expansion	6.4.3		
	power-law graphs	6.4.4		
	$\Delta \leq 3$, partially dynamic	6.7.2		
	$\Delta \leq N^t$	6.4.2	$N^{(1+t)/2-\varepsilon}$	$N^{1+t-\varepsilon}$
(s,t)-distance	$\Delta \leq 3$	6.5.1	$m^{1/2-\varepsilon}$	$m^{1-\varepsilon}$
	$(3-\delta)$-approx, $\Delta \leq 3$	6.5.2		
	constant Δ & expansion	6.5.4		
	power-law graphs	6.5.5		
	$\Delta \leq 3$, partially dynamic	6.7.1		
	$\Delta \leq N^t$	6.5.3	$N^{(1+t)/2-\varepsilon}$	$N^{1+t-\varepsilon}$
Densest Subgraph	$\Delta \leq 5$	6.6.1	$N^{1/4-\varepsilon}$	$N^{1/2-\varepsilon}$
	constant Δ & expansion	6.6.2		
	power-law graphs	6.6.3		

Table 6.2.: Our results for graphs on N nodes with m edges. For every *upd* and *qry* stated in the table, there is no algorithm for the corresponding problem with the corresponding amortized update time and query time simultaneously unless the OMv conjecture is false. The first three rows hold also for perfect matching. All the lower bounds in the table except for densest subgraph match the general OMv lower bounds.

graphs were not only never studied, they were not even defined—removing even a single edge from a power-law graph changes the degree distribution and thus violates the power-law distribution. Hence, we first present several definitions of *dynamic* approximate power-law graphs, where some slackness in the degree-distribution is allowed. Then we prove lower bounds that hold for all of these natural definitions.

6.1.1. Our Results

We consider the maximum matching, (s,t)-distance and densest subgraph problems. We consider the graph classes of constant maximum degree, constant expansion, and power-law degree distribution.

Throughout the chapter we use the standard assumption that queries output one value, such as the size, length or weight of the solution. Note that this makes it only more challenging to prove lower bounds. All our results are conditioned on the popular OMv conjecture [93], but to simplify the terminology we usually drop the word "conditional". Our results are summarized in Table 6.2, where we mention the graph classes the bound applies to.

Main Results. We show the following tradeoff between the update time u and the query time q in an m-edge graph for maximum matching and (s,t)-distance: There is no dynamic algorithm which achieves both $u = O(m^{1/2-\varepsilon})$ and $q = O(m^{1-\varepsilon})$ for any small $\varepsilon > 0$. Note that these bounds match the bounds given for general graphs by Henzinger et al. [93] and that the lower bound for (s,t)-distance is almost tight, as the simple algorithm that only records the edge change at update time and computes the solution from scratch at query time achieves $u = O(1)$ and $q = O(m)$. For densest subgraph we show that there is no dynamic algorithm which achieves both $u = O(N^{1/4-\varepsilon})$ and $q = O(N^{1/2-\varepsilon})$ for any small $\varepsilon > 0$, which is weaker than the lower bound on general graphs (which are quadratically larger).

The only relevant prior work are conditional lower bounds by Abboud et al. [92] for planar graphs, which have constant degree: They show for all-pairs-shortest paths a weaker tradeoff between update time u and query time q than we do, namely they prove $\max(u^2 \cdot q, u \cdot q^2) = \Omega(m^{1-o(1)})$. In *weighted* graphs they show for (s,t)-distance a tradeoff of $\max(u,q) = \Omega(m^{1/2-o(1)})$. Note that our result is stronger as it shows that in *unweighted* graphs no algorithm with $u = \Omega(m^{49/100})$ and $q = \Omega(m^{99/100})$ is possible.

Degree vs Lower Bound Trade-off. While the constant-degree lower bounds are equal to the lower bounds for general graphs in terms of m, they are naturally quadratically lower in terms of the number of nodes N. To understand the behaviour of the bounds also with respect to N, we extend our constant-degree lower bounds for maximum matching, perfect matching, and (s,t)-distance to graphs with maximum degree $O(N^t)$, for any $0 \leq t \leq 1$. We show the following result: There is no dynamic algorithm which achieves both $u = O(N^{(1+t)/2-\varepsilon})$ and $q = O(N^{1+t-\varepsilon})$ in a graph with maximum degree $O(N^t)$ for any $\varepsilon > 0$. These results hold even in bipartite graphs. For $t = 1$, these results exactly match the bounds for general graphs by Henzinger et al. [93], and for $t = 0$, they match our results for constant-degree graphs.

Approximation Results. In constant-degree graphs we extend the lower bound to the problem of $(3-\delta)$-approximating the (s,t)-distance for a small constant δ. This naturally extends the $(3-\delta)$-approximation lower bounds on general graphs to the constant-degree case. The earlier lower bound on planar graphs for computing (s,t)-distance holds only for exact answers.

A similar extension to approximation algorithms is not possible for maximum cardinality matching and for densest subgraph: For maximum matching, for any small $\delta > 0$ the above-mentioned $(1+\delta)$-approximation algorithm [102] achieves an amortized update time of $O(\delta^{-2})$, which is constant for constant δ, thereby precluding any non-trivial lower bounds for approximate maximum matching in the constant-degree setting. Stated differently, *our work shows an interesting dichotomy for dynamic matching matching in constant-degree graphs*: For the exact setting there is no dynamic algorithm which achieves both $u = O(m^{1/2-\varepsilon})$ and $q = O(m^{1-\varepsilon})$ for any small $\varepsilon > 0$, while a $(1+\delta)$-approximation can be achieved in constant time, for any small $\delta > 0$. *The same dichotomy arises for densest subgraph*: For any small $\delta > 0$ there exists a $(1-\delta)$-approximation algorithm with polylogarithmic update time [109], while we show a polynomial lower bound for the exact version.

[109]: Sawlani et al. (2020), "Near-Optimal Fully Dynamic Densest Subgraph"

Partially Dynamic Lower Bounds. We extend the constant Δ bounds for maximum matching and (s,t)-distance also to the partially dynamic setting, achieving the same lower bound as in the fully dynamic setting.

Perfect Matching. A special case of maximum matching is determining whether a perfect matching exists in a bipartite graph. For constant-degree graphs and expander graphs we show the following lower bound: There is no dynamic algorithm which achieves both $u = O(m^{1/2-\varepsilon})$ and $q = O(m^{1-\varepsilon})$ for any small $\varepsilon > 0$. This also extends to the varying-degree setting.

6.2. Related Work

Planar Graphs. Prior work for fully dynamic shortest paths in planar graphs is summarized in Table 6.3 and Table 6.4. There is one further lower bound result in dynamic planar graphs, namely for bipartite maximum weighted matching, showing a tradeoff of $\max\{u,q\} = \Omega(N^{1/2-\varepsilon})$, where u denotes the update time and q the query time [92]. The planar graphs used in these lower bound constructions all have constant degree.

There exists other work on upper bounds in planar graphs. Italiano et al. [97] designed a fully dynamic algorithm for maximum flow and minimum cut with $\tilde{O}(N^{2/3})$ update time in planar graphs. In the decremental setting on

[97]: Italiano et al. (2011), "Improved algorithms for min cut and max flow in undirected planar graphs"

Problem	Assuming	LBs
APSP weighted	APSP conj.	$u \cdot q = \Omega(N^{1-o(1)})$
APSP unit weight	OMv conj.	$\max\{u^2 \cdot q,\ q^2 \cdot u\} = \Omega(N^{1-o(1)})$
(s,t)-distance, girth, diameter	OMv conj.	$\max\{u,\ q\} = \Omega(N^{1/2-\varepsilon})$

Table 6.3.: Lower bounds by Abboud and Dahlgaard [92] for fully dynamic shortest-paths algorithms in planar graphs, where u is the update time and q the query time.

Problem	Ref.	Update	Query
undir. $(1+\varepsilon)$-apx. (s,t)-dist.	[96]	$\tilde{O}(n^{2/3})$	$\tilde{O}(n^{2/3})$
undir. $(1+\varepsilon)$-apx. (s,t)-dist.	[98]	$\tilde{O}(n^{1/2})$	$\tilde{O}(n^{1/2})$
undir. (s,t)-dist., treewidth k	[99]	$O(k^3 \log n)$	$O(k^2 \log n \log(k \log n))$
dir. weighted SSSP	[101]	$\tilde{O}(n^{4/5})$	$O(\log^2 n)$

Table 6.4.: Upper bounds for fully dynamic shortest-path algorithms in planar graphs.

directed graphs, Italiano et al. [100] gave an algorithm with $\tilde{O}(1)$ update time.

[100]: Italiano et al. (2017), "Decremental single-source reachability in planar digraphs"

Other Graph Classes. In $\sqrt{N}$-separable graphs, Goranci et al. [110] gave almost tight bounds for maintaining $(1+\varepsilon)$-approximations of the all-pairs effective resistances. In constant-arboricity graphs, Peleg and Solomon [111] gave $(1+\varepsilon)$-approximate matching algorithm in constant time.

[110]: Goranci et al. (2018), "Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs"

[111]: Peleg et al. (2016), "Dynamic $(1+\epsilon)$-Approximate Matchings: A Density-Sensitive Approach"

Insertions-only and Deletions-only Lower Bounds. In general graphs, Dahlgaard [112] presented lower bounds of $\Omega(N^{1-o(1)})$ for incremental or decremental maximum cardinality bipartite matching, of $\Omega(m^{1-o(1)})$ for incremental or decremental maximum flow in directed and weighted sparse graphs, and $\Omega(N^{1/2-o(1)})$ for partially dynamically $(4/3-\delta)$-approximating the diameter of an unweighted graph for any small constant $\delta > 0$. These lower bounds for diameter were later improved by Ancona et al. [113]. Results for dynamic near-additive spanners were given by Bergamaschi et al. [114].

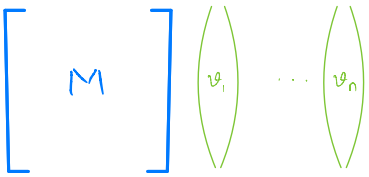
[112]: Dahlgaard (2016), "On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter"

[113]: Ancona et al. (2019), "Algorithms and Hardness for Diameter in Dynamic Graphs"

[114]: Bergamaschi et al. (2021), "New Techniques and Fine-Grained Hardness for Dynamic Near-Additive Spanners"

6.3. Preliminaries

Throughout the chapter, we consider vectors and matrices that are boolean, and so a vector-matrix-vector multiplication outputs a single bit. Henzinger *et al.* [93] define the *Online Matrix Vector* (OMv) and the *Online Vector Matrix Vector* (OuMv) multiplication problems.



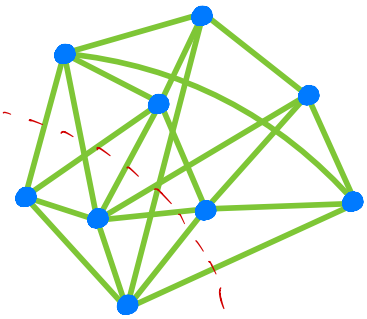
Definition 6.3.1 (Online Matrix Vector Multiplication) *Let M be a boolean $n \times n$ matrix. Preprocessing the matrix is allowed. Then, n vectors $v^1, v^2, \ldots, v^n$ arrive one at a time, and the task is to output the product Mv^i before the next vector is revealed.*

Definition 6.3.2 (Online Vector Matrix Vector Multiplication) *Let M be a boolean $n \times n$ matrix. Preprocessing the matrix is allowed. Then, n vector pairs $(u^1, v^1), (u^2, v^2), \ldots, (u^n, v^n)$ arrive one at a time, and the task is to output the bit $u^i M v^i$ before the next vector pair is revealed.*

They show that the OuMv problem can be reduced to the OMv problem, and conjecture that there is no truly subcubic time algorithm for OMv.

Conjecture 6.3.1 (OMv) *There is no algorithm for the* OMv *(and hence the* OuMv*) problem running in time $O(n^{3-\varepsilon})$ for any constant $\varepsilon > 0$.*

We work with the OuMv problem for all the reductions presented here. We denote the length of our input vectors u^i, v^i by n, and thus the matrix M is of dimension $n \times n$. We use upper indices to indicate the vector's location in the stream, but usually focus on one pair (u, v) omitting these indices. We use lower indices for a location in the vector or matrix, e.g., u_i and M_{ij}. We use N to denote the number of nodes in our reduction graph.



Definition 6.3.3 (Expansion) *The expansion parameter of a graph $G = (V, E)$ is defined as*

$$h = \min\left\{ \frac{|E(S, \bar{S})|}{|S|} \;\middle|\; \emptyset \neq S \subseteq V,\ |S| \leq |V|/2 \right\}$$

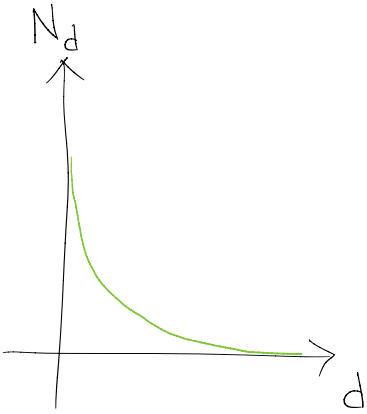
where $E(S, \bar{S})$ is the number of edges from S to $V \setminus S$.

We call a graph with expansion h a *h-expander*.

The parameter above is usually referred to as the *vertex expansion* of a graph. Works on dynamic algorithms use a different definition of expansion, using a parameter h', called *volume expansion*. However, both parameters are within a Δ factor of each other. When considering constant-degree graphs with constant expansion (as we do in this thesis), the difference does not matter, so we only consider the expansion parameter h in our proofs.

[115]: Aiello et al. (2001), "A Random Graph Model for Power Law Graphs"

We study *power-law* graphs as introduced by Aiello et al. [115], and only consider the setting where $\beta > 2$. In the following definition, if the number of nodes N in the graph is fixed, then we get that α is roughly $\ln N$.



Definition 6.3.4 (Power Law Graph) *A graph is said to follow an (α, β)-power law distribution if the number N_d of nodes with degree d is inversely proportional to d^β for some constant $\beta > 0$. That is,*

$$N_d = \left\lfloor \frac{e^\alpha}{d^\beta} \right\rfloor \approx \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor,$$

where $\zeta(\beta) = \sum_{i=1}^{\infty} 1/i^\beta$ is the Riemann Zeta function.

Since dynamic graphs allow edge updates, it is impossible to maintain an exact degree distribution at all times. Hence, we introduce the notion of approximate power-law distributions to afford some slack for dynamic changes. One natural relaxation is to allow β to vary within an interval.

Definition 6.3.5 (β-Varying Power Law) *A graph is said to follow an $(\alpha, \beta_1, \beta_2)$-varying power law distribution if the number N_d of nodes with degree d satisfies*

$$\begin{aligned} &\min\left\{ \left\lfloor \frac{1}{\zeta(\beta_1)} \cdot \frac{N}{d^{\beta_1}} \right\rfloor, \left\lfloor \frac{1}{\zeta(\beta_2)} \cdot \frac{N}{d^{\beta_2}} \right\rfloor \right\} \\ &\leq N_d \\ &\leq \max\left\{ \left\lfloor \frac{1}{\zeta(\beta_1)} \cdot \frac{N}{d^{\beta_1}} \right\rfloor, \left\lfloor \frac{1}{\zeta(\beta_2)} \cdot \frac{N}{d^{\beta_2}} \right\rfloor \right\} \end{aligned}$$

This relaxation of an exact power law, while being natural, is a global relaxation rather than a local one. Thus we also define two locally approximate

definitions below that allow similar slack for all degrees.

Definition 6.3.6 (Additively Approximate Power Law) *A graph is said to follow an (α, β, c)-additively approximate power law distribution if the number N_d of nodes of degree d for a realisable degree d satisfies*

$$\left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor - c \leq N_d \leq \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor + c$$

where we say that d is a realisable degree if there is a node of degree d in an (α, β)-power law graph.

Definition 6.3.7 (Multiplicatively Approximate Power Law) *A graph is said to follow an $(\alpha, \beta, \varepsilon)$-multiplicatively approximate power law distribution if the number N_d of nodes of degree d satisfies*

$$\frac{1}{(1+\varepsilon)} \cdot \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor \leq N_d \leq (1+\varepsilon) \cdot \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor$$

Our lower bounds contain at most four nodes that are one degree away from an exact power-law distribution, and thus hold in all the models discussed above with any reasonable parameter regime.

We note a few useful properties of power-law graphs for our lower bounds. The maximum realizable degree in a power law graph is $\lceil e^{\alpha/\beta} \rceil$, since

$$N_d \geq 1 \iff \left\lfloor \frac{e^\alpha}{d^\beta} \right\rfloor \geq 1 \iff d \leq \lceil e^{\alpha/\beta} \rceil.$$

In terms of N, the maximum degree in a power-law graph is

$$\Delta = \lceil (N/\zeta(\beta))^{1/\beta} \rceil < \sqrt{N}.$$

We work in the setting where $\beta > 2$. Here, the number of edges in the graph is

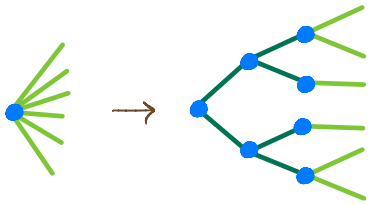
$$|E| = \frac{1}{2} \cdot \frac{\zeta(\beta-1)}{\zeta(\beta)} \cdot N,$$

which is linear in the number of nodes for a fixed β.

6.3.1. Technical Overview

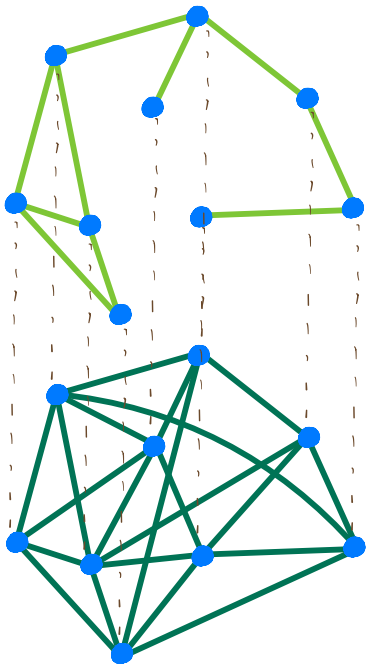
We prove lower bounds by reductions from the online matrix vector (OMv) conjecture [93]. In these reductions, the input of an online problem, which is an $n \times n$ matrix M and a sequence of n pairs (u, v) of n-vectors, is translated into a dynamic graph. The reduction is constructed such that there exists a pair (u, v) satisfying $uMv = 1$ if and only if the dynamic graph has some desired property after a particular sequence of updates. While we follow the general framework of OMv lower bounds, the details are delicate, as the dynamic graphs we construct should fall into specific graph classes at all times, while still maintaining the graph property under consideration. We give a high-level overview of our reductions below.

One way to turn known OMv-to-dynamic graphs reductions into reductions that produce bounded-degree graphs is by replacing high-degree nodes by bounded-degree trees. This technique has a rather clear and straightforward effect on the distances in the graph, so it is applicable when considering distance-related problems. This, however, does not apply when considering other problems such as maximum matchings.



Here, replacing a high-degree node with a gadget could adversely affect the desired matching size, since the gadget might create several augmenting paths that would not have existed when it was a single high-degree node. To overcome this, we limit the possible maximum matching sizes that can arise in our construction, by designing a reduction graph with bounded-degree gadgets composed of paths, where the maximum matching is always either a perfect matching, or a near-perfect matching, i.e., the matching size is either $N/2$ or $N/2 - 1$. This reduction thus involves a large matching and a small gap between the $uMv = 0$ and $uMv = 1$ cases, and hence cannot be extended to achieve a lower bound for the approximation of the maximum matching size. This is a natural barrier. As described above, for any small $\delta > 0$ there is a constant time $(1+\delta)$-approximation dynamic algorithm for the problem, and, thus, such a lower bound cannot exist.

Our reduction for the densest subgraph is more involved. A straightforward reduction would change $O(n)$ graph-edges for every bit of the input, which will allow us to make sure that the density of the densest subgraph changes by a significant amount when $uMv = 0$ versus when $uMv = 1$. However, this would involve $O(n^2)$ updates for each (u, v) input pair, and the reduction would fail to yield any non-trivial lower bound. Thus, we are forced to change very few edges for each input bit, which renders an almost negligible effect on the density, making it difficult to control the exact density of the densest subgraph. Our reduction balances these two factors, using a construction where each gadget is a sufficiently dense regular graph, while having each bit of the input translate into the existence or nonexistence of merely two edges inside specific gadgets. As in the case of matchings, our lower bounds cannot be extended to approximations, as for any $\delta > 0$ there exists a fast algorithm with polylogarithmic update time for computing $(1 - \delta)$-approximations to the densest subgraph.



We then extend these reductions from bounded-degree graphs to constant-degree *constant-expansion* graphs. First, the standard lower bound reductions contain sparse cuts if the inputs M, u or v are sparse, making a standard reduction graph far from being an expander. Thus, we have to augment the graph with many more edges to make sure that it has no sparse cuts regardless of M, u and v. We do this augmentation "inside a layer" to prevent the additional edges from creating undesired short paths between s and t, or spurious augmenting paths in the case of matchings. Sparse cuts also exist in parts of the graph that do not depend on M, u or v, and to handle these, we add edges of a constant-degree expander between a well-chosen set of nodes, thus guaranteeing the expansion without changing the required graph property. Finally, in the case of distance-related problems, we note that expander graphs can have at most logarithmic diameter, but the substitution of nodes by trees described above increases the diameter to be at least logarithmic, leaving only a very small slack for our construction.

When studying densest subgraphs on expanders, adding edges in order to avoid sparse cuts might change the location and structure of the densest subgraph in an undesired way. In order to guarantee expansion, we add a copy of all the graph nodes, build a constant-degree expander on the copies of the nodes, and then connect each node to its copy by a perfect matching.

In dynamic power-law graphs where the node degrees may depend on the inputs u, M, v and change over time, we have to guarantee that the degree changes incurred by the processing of different inputs do not cause a violation of the power-law distribution of degrees. As before, all the changes must also be done without changing the graph property under consideration, and without performing too many update operations. We address this problem by inserting or deleting edges in an online fashion in other parts of the reduction graph, to compensate for the changes incurred by processing the input vector pairs.

6.4. Lower Bounds for Dynamic Maximum Matching

In this section, we present our lower bound results for maximum matching. The previous matching lower bounds on general graphs [93, 112] use reduction graphs that contain nodes with degree $\Omega(N)$. Towards showing a lower bound on expanders, we first sparsify the original reduction.

[93]: Henzinger et al. (2015), "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture"

[112]: Dahlgaard (2016), "On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter"

In Section 6.4.1, we give a lower bound for maximum matching on graphs with maximum degree 3. In Section 6.4.2, we show that the distinction between the unbounded and constant-degree reductions is not discrete, by giving a lower bound reduction parameterized on the maximum degree allowed in the graph. In Section 6.4.3, we give a reduction graph that has constant expansion. Finally, we show our lower bounds for power-law graphs in Section 6.4.4.

6.4.1. Constant-Degree Graph

We first perform a simple reduction that shows that maintaining maximum matchings is hard even on graphs where the maximum degree is 3. We use the following gadget composed of paths to maintain matching properties in our reduction graph during sparsification—see Figure 6.1. Our gadget construction starts by replacing each node of a dense reduction by a path; we refer to each path as a 'subgadget'. Connecting every node of this new subgadget with nodes outside the subgadget might create unwanted matchings of larger sizes, so instead we carefully choose a subset of the path nodes to connect outside the subgadget.

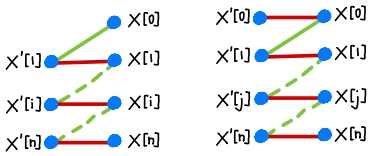


Figure 6.1.: Odd and even sized paths used in the matching lower bounds. The canonical matchings are marked in red.

Consider an odd path on $2n + 1$ vertices, and a bipartition of the vertices into (X', X) with $|X'| \leq |X|$. Indexing the vertices as $X[0]$ and $X'[i], X[i]$ for $1 \leq i \leq n$, our canonical matching matches $X[i]$ with $X'[i]$, leaving $X[0]$ unmatched. We only connect vertices in X outside the subgadget, while vertices in X' (and $X[0]$) only have edges inside the subgadget. For an even path on $2n + 2$ vertices indexed as above, our canonical matching is perfect, and matches $X[i]$ to $X'[i]$. Only $X'[0]$ and all vertices in X are connected outside the subgadget, and the other vertices have edges only within.

Definition 6.4.1 (Reduction gadget) *A reduction gadget with x subgadgets of size y is a bipartite graph composed of x subgraphs, each of which is a path on y nodes.*

Static Graph. The reduction graph is composed of two odd-sized reduction gadgets, and two even-sized reduction gadgets as follows.

- A reduction gadget with one subgadget of size $2n + 1$, on a set $L_1 \cup L_2$. The nodes are labelled $L_1[i]$ and $L_2[i]$ for $0 \leq i \leq n$, and the path is from $L_2[0]$ to $L_2[n]$.
- A reduction gadget with n subgadgets of size $2n + 2$ each, on a set $L_3 \cup L_4$. The subgadgets are labelled $LG[i]$ for $1 \leq i \leq n$, and the nodes of subgadget $LG[i]$ are labelled $L_3[i, j]$ or $L_4[i, j]$ for $0 \leq j \leq n$ depending on whether the node is in L_3 or L_4. The path in each subgadget goes from $L_3[i, 0]$ to $L_4[i, n]$.
- A copy of the above structure, with node sets marked R_1, R_2, R_3, R_4 instead of L_1, L_2, L_3, L_4, respectively.

This reduction graph is bipartite, and so our lower bounds in this section hold for maintaining an exact maximum matching even on bipartite graphs.

The total number of nodes in the reduction graph is therefore

$$N = 4n^2 + 8n + 2 = \Theta(n^2).$$

Input-Dependent Edges.

- If $M_{ij} = 1$, add the edge $(L_4[i, j], R_4[j, i])$.
- If $u_i = 1$, add the edge $(L_2[i], L_3[i, 0])$.
- If $v_j = 1$, add the edge $(R_2[j], R_3[j, 0])$.

Matchings in the Graph. We start by defining a base matching B on the graph, which is made up of the canonical matchings on each of the gadgets. On the left side, B matches $L_3[i, j]$ to $L_4[i, j]$, and $L_1[i]$ to $L_2[i]$ for all i, j. The matching on the right side is similar. Note that this matching always exists regardless of the input, and only $L_2[0]$ and $R_2[0]$ are unmatched in the entire

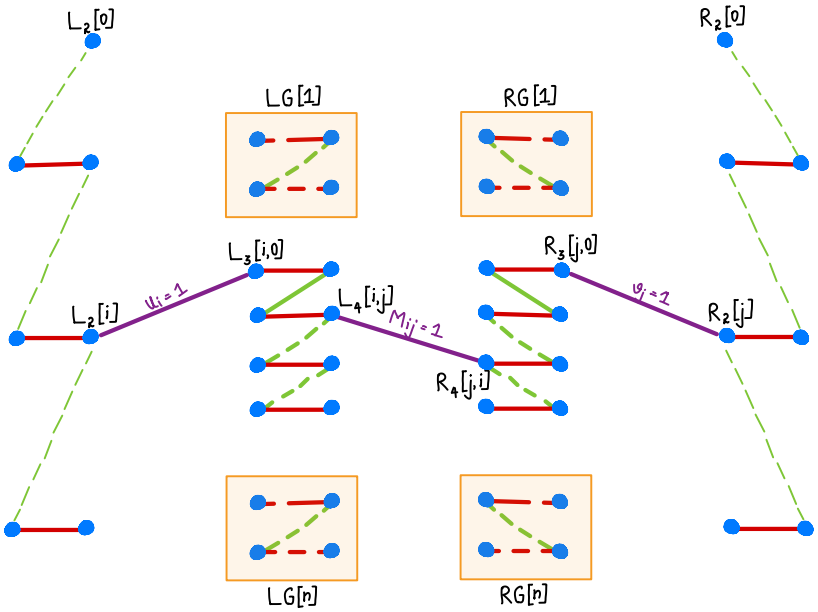


Figure 6.2.: Reduction graph for the matching lower bound. The red edges denote the matching edges, the green edges are the unmatched edges, and the purple edges denote the input-dependent edges.

graph. Thus $|B| = \frac{N}{2} - 1$. We claim that this graph has a perfect matching if and only if $uMv = 1$. Let C denote the maximum cardinality matching.

Lemma 6.4.1 *If $uMv = 1$, then $|C| = \frac{N}{2}$, and otherwise $|C| = \frac{N}{2} - 1$.*

Proof. Since B is always a matching of size $\frac{N}{2} - 1$ regardless of the input, the claim is equivalent to showing that $uMv = 1$ if and only if there is an augmenting path with respect to the matching B.

($\implies$) Suppose that $uMv = 1$, with $u_i = M_{ij} = v_j = 1$. Consider the path P composed of the following subpaths, of which all except P_4 start with an unmatched edge and end with a matched edge, while P_4 both starts and ends with an unmatched edge.

- $P_1 = L_2[0], L_1[1], L_2[1], \dots, L_2[i]$
- $P_2 = L_2[i], L_3[i,0], L_4[i,0], \dots, L_4[i,j]$
- $P_3 = L_4[i,j], R_4[j,i], R_3[j,i], \dots, R_3[j,0]$
- $P_4 = R_3[j,0], R_2[j], R_1[j], \dots, R_2[0]$

Thus, P is an augmenting path to the base matching B, which gives us that the maximum matching C has to have size $> \frac{N}{2} - 1$, implying that the maximum matching C is a perfect matching.

($\impliedby$) Suppose now that there exists an augmenting path P to the base matching B that starts at $s = L_2[0]$ and ends at $t = R_2[0]$. We show that all of M_{ij}, u_i, and v_j are 1.

Matrix M_{ij}: Since $(\cup_i L_i, \cup_j R_j)$ is an (s,t)-cut, there is at least one crossing edge, say $(L_4[i,j], R_4[j,i])$, in P. Thus $M_{ij} = 1$.

Vector u_i: Since P leaves the subgadget $LG[i]$ using $(L_4[i,j], R_4[j,i])$, it should have entered the subgadget at some previous instance. Since the edge $(L_4[i,j], R_4[j,i])$ is unmatched, and all the matching edges in $LG[i]$ are within the subgadget, P should have entered the subgadget using an unmatched edge. As all the matching edges in $LG[i]$ are between L_3 and L_4, P cannot both enter and exit the subgadget through L_4. Thus P enters $LG[i]$ through L_3. However, the only possible unmatched edge from L_3 leaving the subgadget

is the edge $(L_3[i,0], L_2[i])$. Thus P uses the edge $(L_3[i,0], L_2[i])$ to enter the subgadget $LG[i]$, and so $u_i = 1$.

Vector v_i: The path P now enters the subgadget $RG[j]$ through the unmatched edge $(L_4[i,j], R_4[j,i])$. As before, all the matched edges in $RG[j]$ are between R_4 and R_3, and so P has to exit the subgadget using an unmatched edge from R_3. However, the only possible unmatched edge from R_3 leaving the subgadget is the edge $(R_3[j,0], R_2[j])$. Thus the edge $(R_3[j,0], R_2[j])$ is used by P, giving us that $v_j = 1$. □

Complexity of the Reduction. We now prove the theorem.

Theorem 6.4.2 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining a maximum matching or determining the existence of a perfect matching, on all N-node graphs with maximum degree $\Delta \leq 3$, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Proof. The reduction graph above consists of $N = 4n^2+8n+2 = \Theta(n^2)$ nodes. For every new input vector pair (u, v), we delete all the edges between $L_2 \times L_3$ and $R_2 \times R_3$ and insert edges according to the new input vectors. This takes $O(n)$ updates in total. After that, we query once for the size of the maximum matching in this new graph, and return 1 if and only if $|C| = \frac{N}{2}$.

For each pair of input vectors, we perform $O(n)$ updates and $O(1)$ query. In total, checking n pairs takes $O(n^2)$ updates and $O(n)$ query. We can use an algorithm for maximum matching on constant-degree graphs with update time $O(N^{1/2-\varepsilon})$ $(= O(n^{1-2\varepsilon}))$ and query time $O(N^{1-\varepsilon})$ $(= O(n^{2-2\varepsilon}))$ to solve the OMv problem in $O(n^{3-2\varepsilon})$ time, contradicting the OMv conjecture. □

6.4.2. Varying Degree Graph

We present a reduction that gives a lower bound parameterized on the maximum degree of the graph. Note that setting $t = 1$ and $t = 0$ in the following theorem give us the unbounded degree lower bound of Henzinger et al. [93] and Theorem 6.4.2 respectively.

Theorem 6.4.3 *Given any $0 \leq t \leq 1$ and constant $\varepsilon > 0$, there is no dynamic algorithm maintaining a maximum matching or determining the existence of a perfect matching, on all N-node graphs with maximum degree $\Delta = O(N^t)$, with amortized $O(N^{\frac{1+t}{2}-\varepsilon})$ update time and $O(N^{1+t-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Reduction Graph. Given $0 \leq t \leq 1$, we construct a reduction graph that has maximum degree $O(N^t)$.

- L_1, L_2, R_1, R_2 and the edges for u and v are the same as in Section 6.4.1.
- LG is now an even reduction gadget with n subgadgets of size $2n^{\frac{1-t}{1+t}} + 2$ each. The path in each subgadget goes from $L_3[i,0]$ to $L_4[i, n^{\frac{1-t}{1+t}}]$, and similarly for R_3 and R_4.
- If $M_{ij} = 1$, define $i' = \left\lceil i \cdot n^{-2t/(t+1)} \right\rceil$ and $j' = \left\lceil j \cdot n^{-2t/(t+1)} \right\rceil$ and add the edge $(L_4[i,j'], R_4[i',j])$.

Note that the augmenting paths in this reduction graph are the same as in the constant-degree reduction graph by a similar proof as in Lemma 6.4.4. By construction, each node in L_4 is connected to at most $n^{2t/(t+1)}$ nodes in R_4, and each node in R_4 is connected to at most $n^{2t/(t+1)}$ nodes in L_4. The proof of the theorem is similar to the proof of Theorem 6.4.2.

Proof. The number of nodes in the reduction graph described in Section 6.4.2 is dominated by the number of nodes in L_4 and R_4. Thus the total number of nodes in the reduction graph is $N = \Theta(n^{2/(t+1)})$ nodes. Each node in L_4 and R_4 has at most $n^{2t/(t+1)}$ edges of M incident on it by a similar argument as in the proof of Theorem 6.5.5. Thus the maximum degree in the graph is $O(n^{2t/(t+1)}) = O(N^t)$ as required. The rest is similar to Theorem 6.4.2.

Every time we get a new (u, v) input vector pair, we delete all the edges between $L_2 \times L_3$ and $R_2 \times R_3$ and insert edges according to the new input. Thus, for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ queries. In total, checking n pairs takes us $O(n^2)$ updates and $O(n)$ queries. If there were an algorithm for maximum matching on graphs with maximum degree bounded by N^t with update time $O(N^{\frac{1+t}{2}-\varepsilon})$ (i.e., $O(n^{1-2\varepsilon})$) and query time $O(N^{1+t-\varepsilon})$ (i.e., $O(n^{2-2\varepsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-2\varepsilon})$ time, contradicting the OMv conjecture. □

6.4.3. Expander Graph

The previous matching lower bounds on general graphs [93, 112] use reduction graphs that contain nodes with degree $\Omega(N)$. In this section, we construct a constant-degree reduction graph with constant expansion.

Reduction Gadgets. While the reduction gadgets in Section 6.4.1 suffice for sparsification, we need additional constructions in order to guarantee constant expansion. In particular, it turns out that adding edges inside a subgadget does not suffice for constant expansion, and we are forced to add edges between subgadgets. Our construction adds edges on the same side of the bipartition across subgadgets, and our proof implicitly shows that if the newly added edges take part in any augmenting path, then there also exists an augmenting path in the subgraph devoid of any newly inserted edge.

Definition 6.4.2 (Reinforced gadget) *A reinforced gadget with x subgadgets of size y consists of x subgraphs, each of which is a path on y nodes. The nodes are bipartitioned into sets (X', X) with the larger side of the partition labelled as X in each subgadget. Thus $|X'| \leq |X|$. It is then augmented with the following edge-set: Consider a degree-d expander graph on $x \cdot \lceil \frac{y}{2} \rceil$ nodes, choose an arbitrary bijection between the expander nodes and X, and add the expander edges to these nodes accordingly.*

The reduction graph consists of a left subgraph (L) and a right subgraph (R), connected together by edges corresponding to the matrix M. Note that for constant expansion, we need the number of edges of M to be a constant fraction of the sizes of L and R. While it would be possible to construct a reduction graph with $|L|$ and $|R|$ that depend on the input matrix M, we instead choose to augment the input matrix and vectors as it simplifies notation. We thus augment the input beforehand to ensure that there are $\Omega(n^2)$ edges crossing from L to R. To this end, we work with the vectors $\hat{u} = (u\ 0)$ and $\hat{v} = (v\ 0)$ of dimension $2n$, and the matrix $\hat{M} = \begin{pmatrix} M & 1 \\ 1 & 1 \end{pmatrix}$ of dimension $2n \times 2n$. It is easy to see that $uMv = 1 \iff \hat{u}\hat{M}\hat{v} = 1$.

Note that reinforced gadgets are not bipartite. Thus, while the constant-degree lower bounds hold for bipartite matching, the expander result is for maximum matching on general graphs. In what follows, we drop the hats from $\hat{u}, \hat{M}$ and $\hat{v}$ for simplicity, but continue our analysis with their dimensions as $2n, 2n \times 2n, 2n$ respectively.

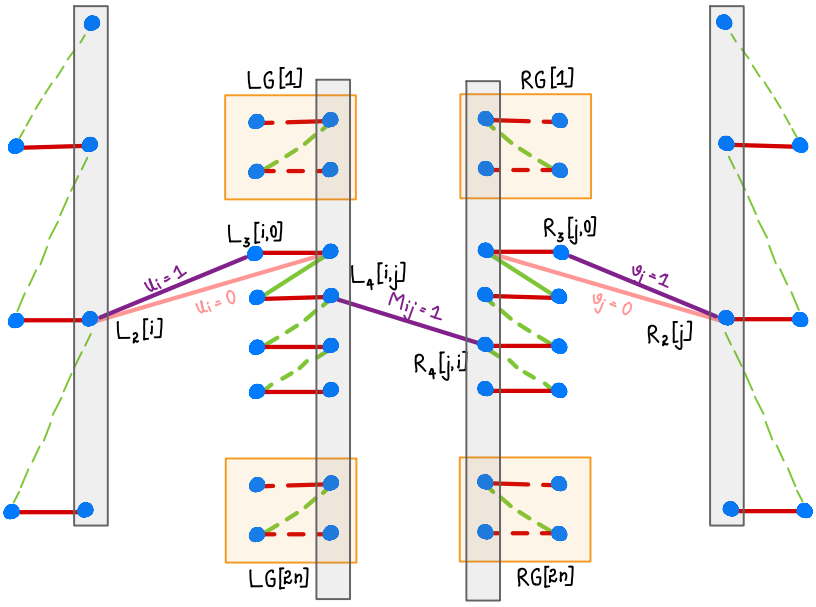


Figure 6.3.: Reduction graph for the expander matching lower bound. It is similar to Figure 6.2, except for the changes due to increasing the dimension, the pink edges when the input is 0, and the grey region which denote the expander edges.

Static Graph. We use the following reduction graph, composed of two odd-sized reinforced gadgets and two even-sized reinforced gadgets.

- A reinforced gadget with one subgadget of size $4n+1$, on a set $L_1 \cup L_2$. The nodes are labelled $L_1[i]$ for $1 \le i \le 2n$, and $L_2[i]$ for $0 \le i \le 2n$. The path is from $L_2[0]$ to $L_2[2n]$, and the expander is on L_2.
- A reinforced gadget with $2n$ subgadgets of size $4n+2$, on a set $L_3 \cup L_4$. The subgadgets are labelled $LG[i]$ for $1 \le i \le 2n$, and the nodes of subgadget $LG[i]$ are labelled $L_3[i,j]$ or $L_4[i,j]$ for $0 \le j \le 2n$ depending on whether the node is in L_3 or L_4. The path in each subgadget goes from $L_3[i,0]$ to $L_4[i,2n]$, and the expander is on L_4.

The total number of nodes in the reduction graph is therefore

$$N = 16n^2 + 16n + 2 = \Theta(n^2).$$

- A copy of the above structure, with node sets marked R_i instead of L_i.

Input-Dependent Edges.

- If $M_{ij} = 1$, add the edge $(L_4[i,j], R_4[j,i])$.
- If $u_i = 1$, add the edge $(L_2[i], L_3[i,0])$.
- If $u_i = 0$, add the edge $(L_2[i], L_4[i,0])$.
- If $v_j = 1$, add the edge $(R_2[j], R_3[j,0])$.
- If $v_j = 0$, add the edge $(R_2[j], R_4[j,0])$.

Matchings in the Graph. Our base matching B is similar to the constant-degree case, and is made up of the canonical matchings on each of the gadgets. On the left side, B matches $L_3[i,j]$ to $L_4[i,j]$, and $L_1[i]$ to $L_2[i]$ for all i, j. The matching on the right side is similar. Note that this matching always exists regardless of the input, and only $L_2[0]$ and $R_2[0]$ are unmatched in the entire graph. Thus $|B| = \frac{N}{2} - 1$. We claim that this graph has a perfect matching if and only if $uMv = 1$. Let C denote the maximum cardinality matching. We show the following lemma.

Lemma 6.4.4 *If $uMv = 1$, then $|C| = \frac{N}{2}$, and otherwise $|C| = \frac{N}{2} - 1$.*

The proof of Lemma 6.4.1 without any modifications shows Lemma 6.4.4 even with the additional edges present.

Complexity of the Reduction. On the arrival of a new vector pair, we first add all the edges corresponding to the new input (if they do not already exist), and then remove the previous vector pair's edges, as opposed to the usual convention of first deleting the previous edges and inserting the new ones. This ensures that the graph remains an expander at all steps. Assuming that we have proved the expansion properties required, since number of edges in a constant-degree graph is $O(N)$, we get the following theorem.

Theorem 6.4.5 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining a maximum matching or determining the existence of a perfect matching, on all N-node graphs with constant degree and constant expansion, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Proof. Consider the reduction graph above. It consists of $N = 16n^2+16n+2 = \Theta(n^2)$ nodes. Every time we get a new (u, v) input vector pair, we update $L_2 \times L_3$ and $R_2 \times R_3$ as detailed above. This takes $O(n)$ updates in total. After that, we query once for the size of the maximum matching in this new graph, and return 1 if and only if $|C| = \frac{N}{2}$.

Thus for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ query. In total, checking n vector pairs takes us $O(n^2)$ updates and $O(n)$ query. If there were an algorithm for maximum matching on constant-degree graphs with update time $O(N^{1/2-\varepsilon})$ (i.e., $O(n^{1-2\varepsilon})$) and query time $O(N^{1-\varepsilon})$ (i.e., $O(n^{2-2\varepsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-2\varepsilon})$ time, contradicting the OMv conjecture. □

Let us now show that the graph described is indeed an h_1-expander graph, for some constant $h_1 > 0$.

Lemma 6.4.6 *The reduction graph has constant expansion.*

Throughout the rest of this section, let S be an arbitrary subset of vertices in the reduction graph, with S satisfying $|S| < N/2 = 8n^2 + 8n + 1$. To simplify our proofs, we consider the reduction graphs with $n > 90$, since then $8n^2 + 8n + 1 < 8.1n^2$. We also use "S *expands*" as a shorthand for $|E(S, \bar{S})| \geq c \cdot |S|$ for some constant $c > 0$.

Note that the node sets in the graph have the following sizes: $|L_1| = 2n$; $|L_2| = 2n + 1$; $|L_3| = |L_4| = 4n^2 + 2n$, and $|R_i| = |L_i|$. We divide the expansion proof into two sections based on whether its intersection with the middle layers (L_4 or R_4) is large or small.

We first deal with the case when the intersection is large.

Lemma 6.4.7 *If $|S \cap L_4| \geq 0.1n^2$ or $|S \cap R_4| \geq 0.1n^2$, then S expands.*

We encapsulate the proof ideas for this lemma in Table 6.5, and split the proof into three lemmas for convenience. First, we show that if its intersection with both the middle layers is very large, then S expands. Our proof uses the canonical matching on $L_3 \cup L_4$.

Lemma 6.4.8 *If $|S \cap L_4| > 3.9n^2$ and $|S \cap R_4| > 3.9n^2$, then S expands.*

Sizes	Proof idea	Proof
$S_{L4} > 3.9n^2 \wedge S_{R4} > 3.9n^2$	Use matching on $L_3 \cup L_4$	Lemma 6.4.8
$0.1n^2 \leq S_{L4} \leq 3.9n^2$	Use expander on L_4	Lemma 6.4.9
$S_{L4} > 3.9n^2 \wedge S_{R4} < 0.1n^2$	Use edges of matrix M	Lemma 6.4.10

Table 6.5.: Ideas for the proof of expansion in the matching reduction graph when either $S \cap L_4$ or $S \cap R_4$ is large. We use S_{L4} as shorthand for $|S \cap L_4|$. The cases when $0.1n^2 \leq S_{R4} \leq 3.9n^2$ and $S_{R4} > 3.9n^2 \wedge S_{L4} < 0.1n^2$ are symmetric to the ones presented here.

Proof. The two conditions together imply that $|S \cap L_3| \leq 0.4n^2$. Thus

$$\begin{aligned}
|E(S, \bar{S})| &\geq |E(S \cap L_4, \bar{S} \cap L_3)| \\
&= |E(S \cap L_4, L_3)| - |E(S \cap L_4, S \cap L_3)| \\
&\geq 3.9n^2 - |E(S \cap L_4, S \cap L_3)| && \text{(by the canonical matching)} \\
&\geq 3.9n^2 - 1.2n^2 && \text{(since } \deg(v) \leq 3 \text{ for all } v \in L_3\text{)} \\
&\geq (2.7/8.1) \cdot |S| && \text{(by upper bound on } |S|\text{)}
\end{aligned}$$

which proves our claim. □

Next, we show that if the intersection with one of the middle layers is of medium size, then S expands. We prove this by using the fact that there is an h_0-expander on the middle layers.

Lemma 6.4.9 *If $0.1n^2 \leq |S \cap L_4| \leq 3.9n^2$, then S expands.*

Proof. Let T be the smaller of the two sets $S \cap L_4$ and $\bar{S} \cap L_4$, then $|T| \leq |L_4|/2$. Thus

$$\begin{aligned}
|E(S, \bar{S})| &\geq |E(S \cap L_4, \bar{S} \cap L_4)| \\
&= |E(T, L_4 \setminus T)| && \text{(by definition of } T\text{)} \\
&\geq h_0 \cdot |T| && \text{(by expansion)} \\
&\geq h_0 \cdot 0.1 \cdot n^2 && \text{(by constraint on } |S \cap L_4|\text{)} \\
&\geq h_0 \cdot (0.1/8.1) \cdot |S| && \text{(by upper bound on } |S|\text{)}
\end{aligned}$$

which proves our claim. □

The above observation also holds when the size of $S \cap R_4$ is in the same range, using the expander on R_4. Finally, we show that if one of the intersections is large and the other is small, then S expands. We use the edges of the matrix M to do this.

Lemma 6.4.10 *If $|S \cap L_4| > 3.9n^2$ and $|S \cap R_4| < 0.1n^2$, then S expands.*

Proof. Recall that our reduction graph has $\geq 3n^2$ edges crossing from L_4 to R_4. Thus

$$\begin{aligned}
|E(S, \bar{S})| &\geq |E(S \cap L_4, \bar{S} \cap R_4)| \\
&= |E(S \cap L_4, R_4)| - |E(S \cap L_4, S \cap R_4)| \\
&\geq |E(S \cap L_4, R_4)| - 0.1n^2 && \text{(by constraint on } |S \cap R_4|\text{)} \\
&= |E(L_4, R_4)| - |E(\bar{S} \cap L_4, R_4)| - 0.1n^2 \\
&\geq 3n^2 - |E(\bar{S} \cap L_4, R_4)| - 0.1n^2 && \text{(by construction)} \\
&\geq 3n^2 - 0.2n^2 - 0.1n^2 && \text{(by constraint on } |S \cap L_4|\text{)} \\
&\geq (2.7/8.1) \cdot |S| && \text{(by upper bound on } |S|\text{)}
\end{aligned}$$

Table 6.6.: Ideas for the first part of the proof of expansion in the left side of the matching reduction graph when $S \cap L_4$ is small ($< 0.1n^2$). Here we deal with the case when either $S_{L4} > 0.1n$ or $S_{L3} > 0.2n$.

Sizes	Proof idea	Proof
$S_{L3} > 2S_{L_4} \wedge S_{L3} > 0.2n$	Use matching on $L_3 \cup L_4$	Lemma 6.4.12
$S_{L3} \leq 2S_{L_4} \wedge S_{L4} > 0.1n$	Use expander on L_4	Lemma 6.4.13

which proves our claim. □

A symmetric argument works by swapping L_4 and R_4 in the above proof. Lemmas 6.4.8, 6.4.9, and 6.4.10 together prove Lemma 6.4.7. We are left with the case when the intersection of S with both the middle layers is small.

Lemma 6.4.11 *If $|S \cap L_4| < 0.1n^2$ and $|S \cap R_4| < 0.1n^2$, then S expands.*

In this case, we show that the intersection of S with the left side ($L = \cup_i L_i$) and the right side ($R = \cup_i R_i$) of the graph expand within their respective sides. This then proves expansion of S, since

$$\begin{aligned} |E(S, \bar{S})| &\geq |E(S \cap L. \bar{S} \cap L)| + |E(S \cap R. \bar{S} \cap R)| \\ &\geq h_1 \cdot |S \cap L| + h_1 \cdot |S \cap R| \quad \text{(by expansion within each side)} \\ &\geq h_1 \cdot |S| \end{aligned}$$

We now concentrate on proving the expansion of $S \cap L$ in L, since the right side expansion follows by similar arguments.

Lemma 6.4.12 *If $|S \cap L_3| > 2|S \cap L_4|$ and $|S \cap L_3| > 0.2n$, then S expands.*

Proof. First, we lower bound the number of crossing edges by $c \cdot |S \cap L_3|$.

$$\begin{aligned} |E(S \cap L, \bar{S} \cap L)| &\geq |E(S \cap L_3, \bar{S} \cap L_4)| \\ &\geq |S \cap L_3| - |S \cap L_4| \quad \text{(by the canonical matching)} \\ &\geq 0.5 \cdot |S \cap L_3| \quad \text{(by constraint on } |S \cap L_4|) \end{aligned}$$

Then we lower bound $22 \cdot |S \cap L_3|$ by $|S \cap L|$, which proves our claim. The final inequality uses the constraint that $|S \cap L_3| > 0.2n$.

$$\begin{aligned} 22 \cdot |S \cap L_3| &= |S \cap L_3| + |S \cap L_3| + 20 \cdot |S \cap L_3| \\ &\geq |S \cap L_3| + |S \cap L_4| + 20 \cdot |S \cap L_3| \quad \text{(by constraint on } |S \cap L_4|) \\ &\geq |S \cap L_3| + |S \cap L_4| + |S \cap (L_1 \cup L_2)| \quad \text{(using } |L_1 \cup L_2| = 4n + 1) \end{aligned}$$

as required. □

Lemma 6.4.13 *If $|S \cap L_3| \leq 2|S \cap L_4|$ and $|S \cap L_4| > 0.1n$, then S expands.*

Proof. Recall that we are working in the case when $|S \cap L_4| < 0.1n^2 < |L_4|/2$.

$$\begin{aligned} |E(S \cap L, \bar{S} \cap L)| &\geq |E(S \cap L_4, \bar{S} \cap L_4)| \\ &\geq h_0 \cdot |S \cap L_4| \quad \text{(by expansion)} \end{aligned}$$

Lower bounding $|S \cap L_4|$ by $c \cdot |S \cap L|$ as in Lemma 6.4.12 gives the claim. □

Sizes	Proof idea	Proof
$S_{L2} \geq 0.7n$	Use edges of u	Lemma 6.4.14
$S_{L2} < 0.7n \wedge S_{L1} \geq 1.3n$	Use matching on $L_1 \cup L_2$	Lemma 6.4.15
$S_{L2} < 0.7n \wedge S_{L1} < 1.3n$	Use gadget expansion	Lemma 6.4.17

Table 6.7.: Ideas for the second part of the proof of expansion in the left side of the matching reduction graph when $S \cap L_4$ is small. Here we deal with the case when $S_{L4} < 0.1n$ and $S_{L3} < 0.2n$.

Now we are only left with the case when $|S \cap L_4| < 0.1n$ and $|S \cap L_3| < 0.2n$. In what follows, we use the bound below on $|S \cap L|$.

$$\begin{aligned} |S \cap L| &= |S \cap (L_1 \cup L_2)| + |S \cap (L_3 \cup L_4)| \\ &\leq 5n + |S \cap (L_3 \cup L_4)| && \text{(by bound on } |L_1 \cup L_2|) \\ &\leq 5.3n && \text{(by constraint on } |S \cap (L_3 \cup L_4)|) \end{aligned}$$

Lemma 6.4.14 *If $|S \cap L_2| > 0.7n$, then S expands.*

Proof. We use the fact that there is an edge from L_2 to $L_3 \cup L_4$ regardless of the value of u.

$$\begin{aligned} |E(S \cap L, \bar{S} \cap L)| &\geq |E(S \cap L_2, \bar{S} \cap (L_3 \cup L_4))| \\ &= |E(S \cap L_2, L_3 \cup L_4)| - |E(S \cap L_2, S \cap (L_3 \cup L_4))| \\ &\geq 0.7n - |E(S \cap L_2, S \cap (L_3 \cup L_4))| && \text{(by edges of } u) \\ &\geq 0.4n && \text{(by constraint on } S \cap (L_3 \cup L_4)) \\ &\geq (0.4/5.3) \cdot |S \cap L| && \text{(by constraint on } S \cap L) \end{aligned}$$

which proves our claim. □

Lemma 6.4.15 *If $|S \cap L_2| < 0.7n$ and $|S \cap L_1| \geq 1.3n$, then S expands.*

Proof. We use the matching on $L_1 \cup L_2$, as follows

$$\begin{aligned} |E(S \cap L, \bar{S} \cap L)| &\geq |E(S \cap L_1, \bar{S} \cap L_2)| \\ &\geq |S \cap L_1| - |S \cap L_2| && \text{(by the canonical matching on } L_1 \cup L_2) \\ &\geq 0.6n && \text{(by constraint on } L_1 \text{ and } L_2) \\ &\geq (0.6/5.3) \cdot |S \cap L| && \text{(by constraint on } S \cap L) \end{aligned}$$

which proves our claim. □

For the final case, we will need the following lemma about each reinforced gadget being an expander.

Lemma 6.4.16 *A reinforced gadget has constant expansion.*

We use this lemma as follows.

Lemma 6.4.17 *If $|S \cap L_2| < 0.7n$ and $|S \cap L_1| < 1.3n$, then S expands.*

Proof. In this case, we reason about $L_1 \cup L_2$ and $L_3 \cup L_4$ separately. Since the intersection of S with each reinforced gadget covers less than half the

nodes, i.e.,

$$|S \cap (L_1 \cup L_2)| < 2n < |L_1 \cup L_2|/2,$$
$$|S \cap (L_3 \cup L_4)| < 0.3n < |L_3 \cup L_4|/2,$$

we use the expansion of each reduction gadget from Lemma 6.4.16 to get

$$|E(S \cap (L_1 \cup L_2), \bar{S} \cap (L_1 \cup L_2))| \geq c \cdot |S \cap (L_1 \cup L_2)|,$$
$$|E(S \cap (L_3 \cup L_4), \bar{S} \cap (L_3 \cup L_4))| \geq c \cdot |S \cap (L_3 \cup L_4)|,$$

giving the expansion of $S \cap L$ as required. □

Lemmas 6.4.12–6.4.17 together prove Lemma 6.4.11. Lemmas 6.4.7 and 6.4.11 together show Lemma 6.4.6. All that is left is to prove Lemma 6.4.16, which we do below.

Lemma 6.4.16 *A reinforced gadget has constant expansion.*

Proof. Let $X' \cup X$ be a reinforced gadget, with the h_0-expander on X. Let S be a subset of nodes of size $< |V|/2$. The proof of expansion follows in similar lines as before.

If the intersection with X is large, then we use the matching edges (similar to Lemma 6.4.8). Concretely, if $|S \cap X| > 0.9|X|$, then since $|X'| \leq |X|$ and $|S| < |V|/2$, we get that $|S \cap X'| < (1/9) \cdot |S \cap X|$. Thus

$$\begin{aligned}
|E(S, \bar{S})| &\geq |S \cap X| - |S \cap X'| && \text{(by the matching on } X' \cup X\text{)} \\
&\geq (8/9) \cdot |S \cap X| && \text{(by constraint on } |S \cap X'|\text{)} \\
&\geq (8/9) \cdot (9/10) \cdot |X| && \text{(by constraint on } |S \cap X|\text{)} \\
&\geq (4/5) \cdot |S| && \text{(since } |X| \geq |V|/2 \text{ and } |S| \leq |V|/2\text{)}
\end{aligned}$$

If the intersection with X is of medium size, then we use the expander on X (similar to Lemma 6.4.9). Concretely, if $0.1|X| \leq |S \cap X| \leq 0.9|X|$, then let T be the smaller of the two sets $S \cap X$ and $\bar{S} \cap X$. Then

$$\begin{aligned}
|E(S, \bar{S})| &\geq |E(S \cap X, \bar{S} \cap X)| \\
&= |E(T, X \setminus T)| && \text{(by definition of } T\text{)} \\
&\geq h_0 \cdot |T| && \text{(by expansion)} \\
&\geq h_0 \cdot 0.1 \cdot |X| && \text{(by constraint on } |S \cap X|\text{)} \\
&\geq h_0 \cdot 0.1 \cdot |S| && \text{(since } |S| \leq |X|\text{)}
\end{aligned}$$

We are left with the case when the intersection with X is small, namely, $|S \cap X| < 0.1|X|$. If $|S \cap X'| > 2|S \cap X|$, then we use the matching edges again.

$$\begin{aligned}
|E(S, \bar{S})| &\geq |S \cap X'| - |S \cap X| && \text{(by the matching on } X' \cup X\text{)} \\
&= (1/3) \cdot (2|S \cap X'| - 4|S \cap X| + |S \cap X'| + |S \cap X|) \\
&\geq (1/3) \cdot (|S \cap X'| + |S \cap X|) && \text{(by constraint on } |S \cap X'|\text{)}
\end{aligned}$$

which leaves us with the final case when $|S \cap X'| \leq 2|S \cap X|$. Here, we use the expander on X

$$\begin{aligned}
|E(S, \bar{S})| &\geq |E(S \cap X, \bar{S} \cap X)| \\
&\geq h_0 \cdot |S \cap X| && \text{(by expansion)} \\
&\geq (1/3) \cdot h_0 \cdot (|S \cap X| + |S \cap X'|) && \text{(by constraint on } |S \cap X'|\text{)}
\end{aligned}$$

which shows that the reinforced gadget has constant expansion. □

6.4.4. Power-law Graph

Static Graph. We first make the reduction graph robust with respect to degree changes.

- A reduction gadget with one subgadget of size $2n+1$, on a set $L_1 \cup L_2$.
- A reduction gadget with $2n$ subgadgets of size $2n+2$, on a set $L_3 \cup L_4$. The subgadgets are labelled $LG[i]$ for $1 \leq i \leq 2n$, and the nodes of subgadget $LG[i]$ are labelled $L_3[i,j]$ or $L_4[i,j]$ for $0 \leq j \leq n$ depending on whether the node is in L_3 or L_4. The path in each subgadget goes from $L_3[i,0]$ to $L_4[i,n]$.
- A copy of the above structure, with node sets marked R_i instead of L_i.

Input-Dependent Edges.

- If $M_{ij} = 1$, add $(L_4[i,j], R_4[j,i])$ and $(L_4[n+i,n+j], R_4[n+j,n+i])$.
- If $M_{ij} = 0$, add $(L_4[i,j], R_4[n+j,n+i])$ and $(L_4[n+i,n+j], R_4[j,i])$.
- The edges for an input pair of vectors (u,v) will be detailed later.

The degree distribution for each layer on the left side of the reduction gadget in the current instance, before adding any edges for (u,v), is given in Table 6.8. For ease of notation, let us use (d, N_d) to denote that there are N_d nodes of degree d in the graph. Thus the degree distribution in the entire reduction graph is as follows: $(1, 4n+2), (2, 4n^2+8n), (3, 4n^2)$. Some of the nodes will change their degree when we add edges for the input vector pair (u,v), and we take care of these changes later.

Table 6.8.: Degree distribution of the nodes on the left side of the reduction graph.

Layer	Deg 1	Deg 2	Deg 3
L_1	0	n	0
L_2	1	n	0
L_3	$2n$	$2n^2$	0
L_4	0	$2n$	$2n^2$

Let $\beta > 2$ be the exponent for our lower bound, and N be the number of nodes we need in our reduction graph. First, choose N such that

$$N > \zeta(\beta) \cdot \max\{(2N_1 + 2n), (2N_2 + 2n) \cdot 2^\beta, (2N_3 + 2n) \cdot 3^\beta\},$$

and pick any power-law graph G on N nodes. If N'_d is the number of nodes of degree d in G, then by construction we get that $N'_d > 2N_d + 2n$.

We would like to essentially embed our reduction graph into this power-law graph. For this, we would like to reduce N'_d by exactly N_d for $d \in \{1,2,3\}$, which would allow us to embed our graph into G. In what follows, we use the fact that $N_1 < N_3$. We "make space" for our nodes as follows.

- Do the following N_1 times: Pick a node $u \in G$ of degree 1. Let w be its neighbour. Since $\deg(w) < \sqrt{N}$ and there are $\geq N'_3 - N_3 > n^2$ nodes of degree 3 in G, there exists a node $v \in G$ of degree 3 such that v has a neighbour $x \in N(v)$ with $(x,w) \notin G$. Delete the edges $(u,w), (v,x)$ and add the edge (x,w). This gives us a node u which we can assign degree 1 to, in our reduction graph. We have also converted a degree 3 node to a degree 2 node, which we take care of in the next step.
- Do the following $N_2 + N_1$ times: Pick a node $u \in G$ of degree 2. Let u_1, u_2 be its neighbours. Since $\deg(u_1) + \deg(u_2) < 2\sqrt{N}$, and there are $\geq N'_2 - N_2$ nodes of degree 2 left in G, there exists a node $v \in G$ of degree 2 that has neighbours v_1, v_2, with $u_i \neq v_j$. Remove the edges $(u,u_i), (v,v_i)$, and add the edge (u_i, v_i). This gives us two nodes u, v which we can assign degree 1 to, in our reduction graph.
- Similarly, free up $N_3 - N_1$ nodes of degree 3 for our reduction graph.

Note that we will have at most one extra node of each degree which we leave unused because of the slack allowed in approximate power-law graphs.

We can now embed our reduction graph into a power-law graph with at most two nodes having different degrees, since there are an even number of nodes of degree 2 in our reduction graph by parity, and we requisitioned degree 1 nodes one-by-one and not in pairs. Thus, we are at most one degree 2 node and one degree 3 node away from a perfect power law graph.

We now come to the question of the input vectors (u, v). If $u_i = 1$, then we add the edge $(L_2[i], L_3[i, 0])$. Note that this changes the degree distribution in the following way: One degree 2 node increases to degree 3, and a degree 1 node increases to degree 2. We need to adjust for this in the power law graph, while making sure that the size of the maximum matching in the remaining graph is still known. We do this as follows:

During preprocessing, pick $2n$ disjoint pairs of nodes (a_i, b_i) in the graph G such that $\deg(a_i) = 2$ and $\deg(b_i) = 3$, such that $\exists\, c_i \in N(a_i), d_i \in N(b_i)$ with $(c_i, d_i) \notin G$. G_0 is the graph G. Let G_j be the graph with the edges $(a_i, c_i), (b_i, d_i)$ deleted and the edges (c_i, d_i) added for all $1 \leq i \leq j$. Since $\text{poly}(n)$ preprocessing is allowed in the OMv conjecture, we can afford to find the sizes of the maximum matchings in all the graphs G_j before we receive any input. Denote the sizes of these matchings as m_j, $0 \leq j \leq 2n$.

On input (u, v), we do the following:

- If $u_i = 1$, add the edge $(L_2[i], L_3[i, 0])$.
- If $v_i = 1$, add the edge $(R_2[i], R_3[i, 0])$.
- Let $k = \text{supp}(u) + \text{supp}(v)$. Delete the edges $(a_i, c_i), (b_i, d_i)$ and add the edges (c_i, d_i) for all $1 \leq i \leq k$.

Now we ask for the size of the maximum matching in this graph. Let $\overline{G}$ be the subgraph of G which is the reduction graph, and let $\overline{N}$ be the number of nodes in $\overline{G}$. Note that we already know the size of the maximum matching in $G \setminus \overline{G}$ to be m_k. $uMv = 1$ if and only if a maximum matching restricted to $\overline{G}$ is perfect, and since $\overline{G}$ is disjoint from $G \setminus \overline{G}$, if and only if the maximum matching on G is of size $m_k + \frac{\overline{N}}{2}$. We then roll back the graph to its previous state and process the new input vector pair.

We make $O(n)$ updates and 1 queries for each input pair, and the graph consists of $\Theta(n^2)$ nodes, which gives us the same lower bounds as in the constant-degree reduction.

6.5. Lower Bounds for (s, t)-Shortest Path

In this section, we present our lower bound results for the dynamic (s, t)-shortest path problem. In Section 6.5.1, we give a lower bound for dynamic (s, t)-distance on graphs with maximum degree 3. We extend this lower bound to $(3 - \delta)$-approximations in Section 6.5.2. In Section 6.5.3, we show that the distinction between the unbounded and constant-degree reductions is not discrete, by giving a lower bound reduction parameterized on the maximum degree allowed in the graph. In Section 6.5.4, we show that the lower bound on constant-degree graphs holds even on expanders, by constructing a more involved reduction graph. Finally, we prove the power-law graph lower bounds in Section 6.5.5.

6.5.1. Constant-Degree Graph

Consider the OuMv problem on vectors of length n and an $n \times n$ matrix. We first perform a simple reduction that shows that maintaining (s,t)-distance is hard even on graphs where the maximum degree is 3.

Theorem 6.5.1 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining (s,t)-distance, SSSP or APSP, on all N-node bipartite graphs with maximum degree $\Delta \leq 3$, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Since the original reduction [93] could possibly have unbounded degree, we use *binary forests* to sparsify our reduction graph.

Definition 6.5.1 (Binary Forest) *A binary forest of x trees of height y is a graph composed of x disjoint binary trees, each of height y.*

We naturally split the nodes of a binary forest between *internal nodes* and *leaves*. Intuitively, we would like to replace a high degree node of the original reduction with a binary forest to moderate the maximum allowed degree. Note that a binary forest has $x \cdot 2^y$ leaves in total.

Static Graph. We use the following static graph for our reduction.

- A $(\log n)$-depth binary forest with a single tree. The set of $n-1$ internal nodes is marked L_1 (L for left) and the n leaves L_2; the root of the tree is the source node s, and the n nodes of L_2 are marked as $L_2[i]$ for $1 \leq i \leq n$.
- A $(\log n)$-depth binary forest with n trees. The $n(n-1)$ internal nodes are marked L_3 and the n^2 leaves L_4. The roots of each of the n trees are marked as $L_3[i]$, for $1 \leq i \leq n$, and the leaves of the tree with root $L_3[i]$ are marked $L_4[i,j]$, for $1 \leq j \leq n$.
- A copy of the above structure, with node sets marked R_1, R_2, R_3, R_4 instead of L_1, L_2, L_3, L_4, respectively. The root of the single tree of R_1 is the target node t.

The total number of nodes in the reduction graph is therefore

$$N = 4n^2 + 2n - 2 = \Theta(n^2).$$

Input-Dependent Edges. We add the following edges depending on the input matrix M and vectors u, v:

- If $M_{ij} = 1$, add the edge $(L_4[i,j], R_4[j,i])$.
- If $u_i = 1$, add the edge $(L_2[i], L_3[i])$.
- If $v_j = 1$, add the edge $(R_2[j], R_3[j])$.

Distances in the Graph. We now show the correctness of the reduction, by considering the (s,t) distance in different scenarios.

Lemma 6.5.2 (constant-degree reduction) *It holds that $uMv = 1$ if and only if $\text{dist}(s,t) \leq 4\log n + 3$. Moreover, the graph is bipartite.*

Proof. We first show that any path from s to t has to be of length at least $4\log n + 3$, by partitioning the node set into layers. We maintain the property that a node at level ℓ can have neighbours only in levels $\ell - 1$, ℓ, or $\ell + 1$. The layering is as follows: The layer of a node in $L_1 \cup L_2$ is its distance from s; in $L_3 \cup L_4$ is its distance from its closest root, plus $\log n + 1$; in $R_3 \cup R_4$ is its distance from its closest leaf, plus $2\log n + 2$; in $R_1 \cup R_2$ is its distance from its closest leaf, plus $3\log n + 3$. Specifically, the layer of node t is $4\log n + 3$. Thus $\text{dist}(s,t) \geq 4\log n + 3$, regardless of u, M, and v. Bipartiteness follows from the fact that edges only connects consecutive layers.

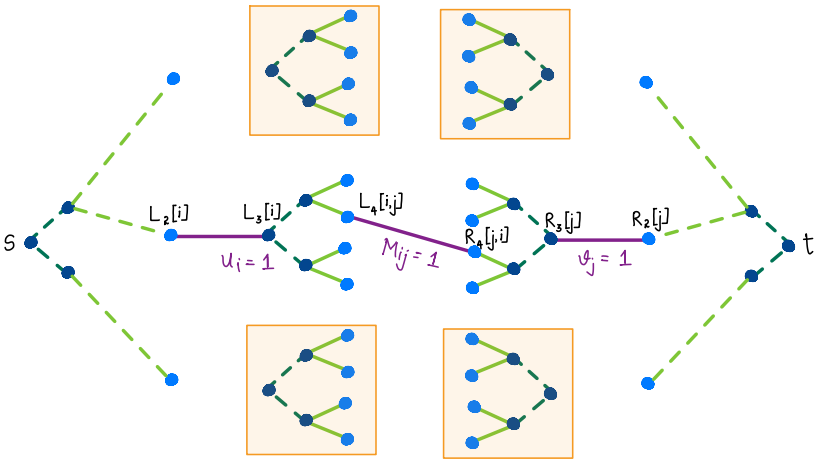


Figure 6.4.: Reduction graph for the (s,t)-shortest path lower bound. The dark green edges denote the internal forest edges, the light green edges denote the leaf edges, and the purple edges denote the input-dependent edges.

($\Longrightarrow$) If $uMv = 1$, then there exists indices i, j such that $u_i = M_{ij} = v_j = 1$. Then consider the path P composed of the following sub-paths:

- $\log n + 1$ edges: P_1 is the shortest path from s to $L_3[i]$, which follows the tree $L_1 \cup L_2$ and then the $(L_2[i], L_3[i])$ edge.
- $\log n + 1$ edges: P_2 is the shortest path from $L_3[i]$ to $R_4[j,i]$, which follows the i^{th} tree rooted at $L_3[i]$ and then the $(L_4[i,j], R_4[j,i])$ edge.
- $\log n + 1$ edges: P_3 is the shortest path from $R_4[j,i]$ to $R_2[j]$, which follows the j^{th} tree rooted at $R_3[j]$ and then the $(R_3[j], R_2[j])$ edge.
- $\log n$ edges: P_4 is the shortest path from $R_2[j]$ to t, which follows the tree $R_1 \cup R_2$.

P is then a path of length $4\log n + 3$ from s to t.

($\Longleftarrow$) Assume that there is a path of length $4\log n + 3$ from s to t. By the layering, each edge in the path must connect a layer ℓ node and a layer $(\ell+1)$ node, and there is exactly one such edge in the path for each $\ell \in [4\log n + 2]$. Next, we use these two facts (sometimes implicitly) to show that the path must have the form of the above described path, and conclude that $uMv = 1$.

The path must contain an edge from L_2 to L_3. The only such edges are of the form $(L_2[i], L_3[i])$ for some $i \in [n]$, implying $u_i = 1$. From there, the path must continue to some leaf $LU[i,j]$ of the tree rooted at $LU[i]$. Since the only edge from $L_4[i,j]$ that strictly increases in level is the edge $(L_4[i,j], R_4[i,j])$, the path must contain such an edge, implying $M_{ij} = 1$ for some $j \in [n]$. From $R_4[j,i]$, the path must continue to the root $R_3[j]$, and then to $R_2[j]$ over an edge $(R_3[j], R_2[j])$, implying $v_j = 1$. The path ends trivially by going from $R_2[j]$ to the root t. Thus $uMv = 1$. □

Complexity of the Reduction. We now prove the theorem.

Theorem 6.5.1 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining (s,t)-distance, SSSP or APSP, on all N-node bipartite graphs with maximum degree $\Delta \leq 3$, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Proof. Consider the reduction graph above, which is bipartite by Lemma 6.5.2. It consists of $N = 4n^2 + 2n - 2 = \Theta(n^2)$ nodes. Every time we get a new (u, v) input vector pair, we delete all the edges between $L_2 \times L_3$ and $R_2 \times R_3$ and insert edges according to the new input vectors. This takes $O(n)$ updates in total. After that, we query once for the (s,t)-distance in this new graph, and return 1 if and only if $\text{dist}(s,t) = 4\log n + 3$.

Thus for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ query. In total, checking n pairs takes us $O(n^2)$ updates and $O(n)$ query. If there were an algorithm for (s,t)-distance on constant-degree graphs with update time $O(N^{1/2-\varepsilon})$ (i.e., $O(n^{1-2\varepsilon})$) and query time $O(N^{1-\varepsilon})$ (i.e., $O(n^{2-2\varepsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-2\varepsilon})$ time, contradicting the OMv conjecture. □

6.5.2. $(3-\delta)$-Approximation Lower Bound for Constant-Degree Graphs

We show that the above lower bounds on (s,t)-distances also holds for $(3-\delta)$-approx (s,t)-distances by minimally modifying the reduction graph to exploit the following observation: If $uMv = 0$, then every path from s to t in the simple reduction graph needs to take at least three edges corresponding to the matrix M, as opposed to just one such edge when $uMv = 1$.

We use the same reduction graph as before, but with one important difference: Earlier, we added a single edge $(L_4[i,j], R_4[j,i])$ if $M_{ij} = 1$. Now, we add a path with $\Theta(\log n)$ new nodes between $L_4[i,j]$ and $R_4[j,i]$. Since we only do this for M and not for every new input vector pair, we can do this in polynomial pre-processing time, which is allowed for in the OMv conjecture.

Formally, let $\alpha = \left\lceil \frac{12}{\delta} - 4 \right\rceil$. Add $n^2 \cdot \alpha \log n$ new nodes to the reduction graph, with the nodes labelled $v_{ij}[k]$ for $1 \le k \le \alpha \log n$ and $1 \le i,j \le n$. If $M_{ij} = 1$, add the path $L_4[i,j], v_{ij}[1], \ldots, v_{ij}[\alpha \log n], R_4[i,j]$ to the reduction graph, while otherwise all the nodes $v_{ij}[k]$ remain disconnected.

The following lemma is an analogue of Lemma 6.5.2, and the proof follows from the proof of Lemma 6.5.2 and the above observation.

Lemma 6.5.3 *If $uMv = 1$, then $\mathrm{dist}(s,t) = (4+\alpha)\log n + 2$, and otherwise $\mathrm{dist}(s,t) \ge (4+3\alpha)\log n + 2$. Moreover, the graph is bipartite.*

Corollary 6.5.4 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining $(3-\delta)$-approximate (s,t)-distance, SSSP or APSP, on all N-node bipartite graphs with maximum degree $\Delta \le 3$, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the OMv conjecture is false.*

Proof. From Lemma 6.5.3, any approximate reported distance d where $(4+\alpha)\log n + 2 \le d < (4+3\alpha)\log n + 2$ would imply that $uMv = 1$. Note that

$$\begin{aligned}
\frac{(4+3\alpha)\log n + 2}{(4+\alpha)\log n + 2} &= 3 - \frac{8\log n + 4}{(4+\alpha)\log n + 2} \\
&\ge 3 - \frac{12\log n}{(4+\alpha)\log n} && \text{(since } \log n \ge 1\text{)} \\
&\ge 3 - \delta && \text{(by definition of } \alpha\text{)}
\end{aligned}$$

Thus any $(3-\delta)$-approx algorithm would be able to distinguish between $uMv = 1$ and $uMv = 0$.

The number of nodes in the reduction graph is $N = \Theta(n^2 \log n) = O(n^{2+\varepsilon})$. Thus for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ queries. In total, checking n pairs takes us $O(n^2)$ updates and $O(n)$ queries. If there were an algorithm for $(3-\delta)$-approx (s,t)-distance on constant-degree graphs with update time $O(N^{1/2-\varepsilon}) = O(n^{1-3\varepsilon/2-\varepsilon^2})$ and query time

$O(N^{1-\varepsilon}) = O(n^{2-\varepsilon-\varepsilon^2})$, then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-c})$ time for some constant c, contradicting the OMv conjecture. □

6.5.3. Varying-Degree Graph

We present a reduction that gives a lower bound parameterized on the maximum degree in the graph.

Theorem 6.5.5 *Given any $0 \leq t \leq 1$ and constant $\varepsilon > 0$, there is no dynamic algorithm maintaining (s,t)-distance, SSSP or APSP, on all N-node bipartite graphs with maximum degree $\Delta = O(N^t)$, with amortized $O(N^{\frac{1+t}{2}-\varepsilon})$ update time and $O(N^{1+t-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Reduction Graph. For a value $0 \leq t \leq 1$ (that may depend on N), we construct a reduction graph on N nodes that has maximum degree $\Theta(N^t)$. The node sets L_1, L_2, R_1, R_2 and the edges that depend on u and v are the same as in the previous construction in Section 6.5.1. We detail the changes for L_3, L_4, R_3, R_4 and M below.

- A $\left(\frac{1-t}{1+t} \cdot \log n\right)$-depth binary forest with n trees. The $n \cdot \left(n^{\frac{1-t}{1+t}} - 1\right)$ internal nodes are marked L_3 and the $n \cdot n^{\frac{1-t}{1+t}}$ leaves are marked L_4. The roots of each of the n trees are marked as $L_3[i]$, for $1 \leq i \leq n$, and the leaves of the tree with root $L_3[i]$ are marked $L_4[i,j]$, for $1 \leq j \leq n^{\frac{1-t}{1+t}}$, and similarly for R_3 and R_4.
- For the matrix M, if $M_{ij} = 1$, define $i' = \left\lceil i \cdot n^{-2t/(t+1)} \right\rceil$ and define $j' = \left\lceil j \cdot n^{-2t/(t+1)} \right\rceil$. Add the edge $(L_4[i,j'], R_4[i',j])$ to the graph.

Note that the distances in this reduction graph are the same as in the constant-degree reduction graph by a similar proof as used to prove Lemma 6.5.2. Furthermore, each node in L_4 is connected to at most $n^{2t/(t+1)}$ nodes in R_4, and each node in R_4 is connected to at most $n^{2t/(t+1)}$ nodes in L_4.

We can now prove Theorem 6.5.5.

Proof. The number of nodes in the reduction graph above is dominated by the number of nodes in L_4 and R_4. Thus, the total number of nodes in the reduction graph is $N = \Theta(n^{2/(t+1)})$ nodes. Since each node not in L_4 or R_4 have at most 3 edges adjacent on it, its degree is trivially $O(N^t)$. Every node in L_4 and R_4 has one tree edge incident on it, and at most $n^{2t/(t+1)}$ edges of M incident on it by construction. Thus the maximum degree in the graph is $O(n^{2t/(t+1)}) = O(N^t)$ as claimed. The rest is similar to Theorem 6.5.1. Every time we get a new (u,v) input vector pair, we delete all the edges between $L_2 \times L_3$ and $R_2 \times R_3$ and insert edges according to the new input vectors.

For each pair of input vectors, we thus perform $O(n)$ updates and $O(1)$ queries. In total, checking n pairs takes $O(n^2)$ updates and $O(n)$ queries. If there was an algorithm for (s,t)-distance on graphs with maximum degree bounded by N^t with update time $O(N^{\frac{1+t}{2}-\varepsilon})$ (i.e., $O(n^{1-2\varepsilon})$) and query time $O(N^{1+t-\varepsilon})$ (i.e., $O(n^{2-2\varepsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-2\varepsilon})$ time, contradicting the OMv conjecture. □

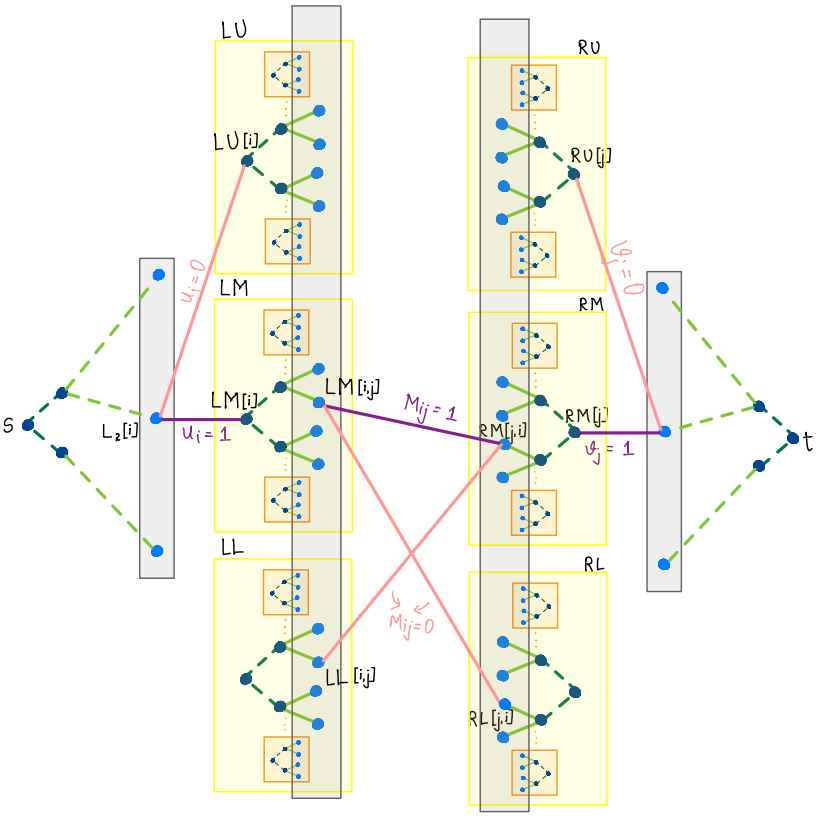


Figure 6.5.: Reduction graph for the expander (s,t)-shortest path lower bound. It is similar to Figure 6.4, except for the changes due to increasing the dimension, the pink edges when the input is 0, and the grey region which denote the expander edges.

6.5.4. Expander Graph

For our expander lower bound, we use *reinforced forests*. We naturally split the nodes of a reinforced forest between *internal nodes* and *leaves*.

Static Graph. We use the following static graph for our reduction.

- A $(\log n)$-depth reinforced forest with a single tree. The set of $n-1$ internal nodes is marked L_1 and the n leaves L_2; the root of the tree is the source node s, and the n nodes of L_2 are marked as $L_2[i]$ for $1 \le i \le n$.
- A $(\log n)$-depth reinforced forest with $3n$ trees. The $3n(n-1)$ internal nodes are marked L_3 and the $3n^2$ leaves L_4. We split this reinforced forest into sets of n trees each, and label them LU, LM, LL (upper, middle, and lower). The roots of each of the n trees are marked as $LX[i]$, for $1 \le i \le n$, $X \in \{U, M, L\}$, and the leaves of the tree with root $LX[i]$ are marked $LX[i,j]$, for $1 \le j \le n$.
- A copy of the above structure, with node sets marked R_i instead of L_i, respectively. The root of the single tree of R_1 is the target node t.

Definition 6.5.2 (Reinforced forest) *A reinforced forest of x trees of height y is a graph composed of x disjoint binary trees, each of them of height $y > 0$. These trees have $x2^y$ leaves in total; consider a degree-d expander graph on $x2^y$ nodes, choose an arbitrary bijection between the expander nodes and the forest's leaves, and add the expander edges to the leaves accordingly. Finally, in order to guarantee that the graph is bipartite, on each edge added from the expander, we add a dummy node.*

Input-Dependent edges.

- If $M_{ij} = 1$, add the edge $(LM[i,j], RM[j,i])$.
- If $M_{ij} = 0$, add the edges $(LM[i,j], RL[j,i])$ and $(LL[i,j], RM[j,i])$.
- If $u_i = 1$, add the edge $(L_2[i], LM[i])$.
- If $u_i = 0$, add the edge $(L_2[i], LU[i])$.
- If $v_j = 1$, add the edge $(R_2[j], RM[j])$.
- If $v_j = 0$, add the edge $(R_2[j], RU[j])$.

The following lemma characterizes (s,t)-distance in the expander reduction graph based on the value of uMv. We split the graph nodes into layers and use the fact that edges can only connect consecutive layers in order to prove this lemma in Section 6.5.4.

Lemma 6.5.6 *It holds that $uMv = 1$ if and only if $\text{dist}(s,t) \leq 4\log n + 3$. Moreover, the graph is bipartite.*

The number of nodes in the reduction graph is dominated by the nodes in $L_4 \cup R_4$, yielding the same asymptotic lower bounds as in the constant-degree case (Section 6.5.1). When a vector pair arrives, we first add all the potential edges and then remove the unnecessary ones, in a similar fashion as before, to preserve expansion. We defer the proof of expansion to Section 6.5.4. Thus, we get the following theorem for expander graphs.

Theorem 6.5.7 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining (s,t)-distance, SSSP or APSP, on all N-node constant-degree bipartite graphs with constant expansion, with amortized $O(N^{1/2-\varepsilon})$ update time and $O(N^{1-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Distances in the Expander Graph. We prove the following lemma for the expander reduction graph described in Section 6.5.4. This is basically the graph described in Section 6.5.1, with expander edges added in carefully chosen places, and one dummy node added on each such edge. Note that adding the dummy nodes does not change the expansion by more than a constant factor, does not affect the asymptotic number of nodes, and the shortest paths never use the expander edges and thus they also stay unaffected by the dummy nodes addition. To simplify the writing, we thus ignore the dummy nodes in the following.

Lemma 6.5.6 *It holds that $uMv = 1$ if and only if $\text{dist}(s,t) \leq 4\log n + 3$. Moreover, the graph is bipartite.*

Proof. The proof uses the same ideas and layering as in the reduction in Lemma 6.5.2. We include the dummy nodes of L_2 and L_4 in one layer before their (non-dummy) neighbors, and the other dummy nodes in one layer after their neighbors. This already shows that the graph is bipartite.

($\Longrightarrow$) This direction of the proof is the same as in the proof of Lemma 6.5.2, since we only add edges, and the same path as earlier still exists in the graph.

($\Longleftarrow$) We still make use of the layering argument for this direction, but the argument is slightly more nuanced than the one for the constant-degree case. Assume that there is a path of length $4\log n + 3$ from s to t. By the layering, each edge in the path must connect a layer ℓ node and a layer $(\ell+1)$ node, and there is exactly one such edge in the path for each $\ell \in [4\log n + 2]$. Next, we use these two facts (sometimes implicitly) to show that the path must have the form of the above described path, and conclude that $uMv = 1$.

- The path must contain an edge from L_2 to L_3. Suppose the edge was to a node in LU. Since no node in LU has an edge to R_4, the path does not always connect two nodes of strictly increasing layers, contradicting the claimed length. Thus the edge is of the form $(L_2[i], LM[i])$ for some $i \in [n]$, implying that $u_i = 1$.

- From there, the path must continue to some leaf $LM[i,j]$ of the tree rooted at $LM[i]$. Suppose the path continues to RL instead of RM. Since there are no edges from RL to R_2, the path would have to connect two of the non-increasing at least once, which cannot happen on a $4\log n+3$ length path. Thus the path uses the edge $(LM[i,j], RM[j,i])$, implying that $M_{ij}=1$.
- From $RM[j,i]$, the path must continue to the root $RM[j]$, and then to $R_2[j]$ by using the edge $(RM[j], R_2[j])$, implying $v_j=1$.

We have established that there exist indices $i,j\in[n]$ such that $u_i = M_{ij} = v_j = 1$, implying $uMv = 1$. □

Expansion of the expander graph. Let us now verify that the graph is indeed an h_1-expander graph, for some constant $h_1>0$. Recall that the reinforced forests contain h_0 expanders, for some constant $h_0>0$.

The proof is similar to that of the maximum matching expander reduction, and we give a less structured presentation here.

Consider a reinforced forest with a set U of internal nodes, a set U' of leaves, and an h_0-expander on the leaves. Let S be a set of nodes in the forest (we do not bound $|S|$).

Observation 6.5.1 *Let $0<c<1/2$, that may depend on n. If $c\cdot|U'| \le |S\cap U'| \le (1-c)\cdot|U'|$, then $E(S,\bar{S})| \ge c\cdot h_0\cdot|U'|$.*

Proof. Consider first the case $|S\cap U'|\le |U'|/2$. By expansion of U',

$$\begin{aligned} |E(S,\bar{S})| &\ge |E(S,\bar{S})\cap U'| \\ &\ge h_0\cdot|S| && \text{(by expander edges on } U') \\ &\ge c\cdot h_0\cdot|U'| && \text{(by assumption on } |S\cap U'|) \end{aligned}$$

Otherwise, apply the analogous argument on the set $U'\setminus S$. □

Observation 6.5.2 $|E(S,\bar{S})| \ge |U\cap S| - |U'\cap S|$.

Proof. Every node in $U\cap S$ is a tree node that has two children. In total, these node has $2|U\cap S|$ edges going to children, of which at most $|U\cap S| + |U'\cap S|$ children are in S. Hence,

$$|E(S,\bar{S})| \ge 2|U\cap S| - (|U\cap S| + |U'\cap S|) = |U\cap S| - |U'\cap S|.$$ □

Observation 6.5.3 *Let $0<c<1/2$ constant. If $|U\cap S| \le c\cdot|U'\cap S|$ then $|E(S,\bar{S})| \ge (0.5-c)\cdot|U'\cap S|$.*

Proof. If $|U\cap S|\le c\cdot|U'\cap S|$, consider the parent nodes of the leaves in $U'\cap S$: these have at least $0.5\cdot|U'\cap S|$ distinct parents, of which at most $c\cdot|U'\cap S|$ are in $U\cap S$, and the others are in $U\cap\bar{S}$. Hence

$$|E(U\cap\bar{S}, U'\cap S)| \ge (0.5-c)\cdot|U'\cap S|.$$ □

The proof of the following lemma is similar to that of Lemma 6.4.16.

Lemma 6.5.8 *A reinforced forest is an expander.*

Fix a set S of nodes in the forest, $|S| \leq (|U \cup U'|)/2$. We consider different ranges of $|U' \cap S|$, and show that for each of them, $|E(S, \bar{S})| \geq h_1|S|$ for some constant h_1. Lemma 6.5.8 follows from the following three claims.

Claim 6.5.9 *If $|U' \cap S| > 0.9 \cdot |U'|$, then S expands.*

Proof. In this case, since $|U| < |U'|$ and $|S| \leq |U \cup U'|/2$, we have that $|U \cap S| < 0.1 \cdot |U'|$. Thus

$$|U' \cap S| > (1/9) \cdot |U \cap S| > 0.1 \cdot |U \cap S|,$$

and we can apply Observation 6.5.3. We have that

$$\begin{aligned}
|E(S, \bar{S})| &\geq 0.4 \cdot |U' \cap S| && \text{(by Observation 6.5.3)} \\
&\geq 0.36 \cdot |U'| && \text{(by assumption)} \\
&> 0.18 \cdot |U \cup U'| && \text{(since } |U| \leq |U'|) \\
&\geq 0.36 \cdot |S| && \text{(since } |S| \leq |U \cup U'|/2)
\end{aligned}$$

as required. □

Claim 6.5.10 *If $0.1 \cdot |U'| \leq |U' \cap S| \leq 0.9 \cdot |U'|$, then S expands.*

Proof. We have

$$\begin{aligned}
|E(S, \bar{S})| &\geq 0.1 \cdot h_0 \cdot |U'| && \text{(by Observation 6.5.1)} \\
&\geq 0.05 \cdot h_0 \cdot |U \cup U'| && \text{(since } |U| \leq |U'|) \\
&\geq 0.1 \cdot h_0 \cdot |S| && \text{(since } |S| \leq |U \cup U'|/2)
\end{aligned}$$

as required. □

Claim 6.5.11 *If $|U' \cap S| < 0.1 \cdot |U'|$, then S expands.*

Proof. We first deal with the setting when $|U \cap S| > 2 \cdot |U' \cap S|$. In this case, add $0.5 \cdot |U \cap S| - 1.5 \cdot |U' \cap S|$ to both sides of the inequality and multiply by $2/3$ to get

$$|U \cap S| - |U' \cap S| > 1/3 \cdot (|U \cap S| + |U' \cap S|) = |S|/3.$$

Then we get

$$|E(S, \bar{S})| \geq |U \cap S| - |U' \cap S| > |S|/3. \qquad \text{(by Observation 6.5.2)}$$

Else, we have $|U \cap S| \leq 2|U' \cap S|$. In this case,

$$|S| = |U \cap S| + |U' \cap S| \leq 3 \cdot |U' \cap S|.$$

Then we get

$$|E(S, \bar{S})| \geq h_0 \cdot |U' \cap S| \geq (1/3) \cdot h_0 \cdot |S| \qquad \text{(by expansion of } U')$$

as required. □

Lemma 6.5.12 *The reduction graph is an expander.*

Consider a set $S \subseteq V$ of $|S| \leq N/2 = 6n^2 - n - 1$ nodes. Note that the node sets in the graph have the following sizes:

$$|L_1| = |R_1| = n - 1 \qquad |L_2| = |R_2| = n$$
$$|L_3| = |R_3| = 3n^2 - 3n \qquad |L_4| = |R_4| = 3n^2.$$

We show that there exists a constant h_1 such that $|E(S, \bar{S})| \geq h_1|S|$, by considering different ranges for the set size $|L_4 \cap S|$ (which ranges in $0, \ldots, 3n^2$). In most cases, we show that $|E(S, \bar{S})| \geq cn^2$ for some constant c, which is enough since $|S| < 6n^2$.

The first case is when $L_4 \cap S$ has a large size. In this case, we deal with different cases based on the size of $R_4 \cap S$.

Lemma 6.5.13 *If $|S \cap L_4| \geq 2.9 \cdot n^2$, then S expands.*

Proof. We look at the size of $R_4 \cap S$. The first case is when it is large, i.e., $|R_4 \cap S| > 2.1n^2$. Since $|S| \leq 6n^2 - n - 1$, we have that $|L_3 \cap S| < n^2$, which implies $|L_4 \cap S| > 2.9 \cdot |L_3 \cap S|$. Thus we have that

$$\begin{aligned} |E(S, \bar{S})| &\geq 0.15 \cdot |L_4 \cap S| && \text{(by Observation 6.5.3)} \\ &> 0.4 \cdot n^2 \end{aligned}$$

as required. The second case is when the size is medium, i.e., $0.1 \cdot n^2 \leq |R_4 \cap S| \leq 2.1 \cdot n^2$. The expander edges on R_4 guarantee

$$E(S, \bar{S})| \geq 0.03 \cdot h_0 \cdot n^2 \qquad \text{(by Observation 6.5.1)}$$

The final case is when the size is small, i.e., $|R_4 \cap S| < 0.1 \cdot n^2$. Note that there are n^2 edges in $L_4 \times R_4$, of which at most $0.1n^2$ are in $(L_4 \cap \bar{S}) \times (R_4 \cap \bar{S})$ (since $L_4 \cap S$ is large), and at most $0.1n^2$ are in $(L_4 \cap S) \times (R_4 \cap S)$ (since $R_4 \cap S$ is small). Thus, the remaining edges, at least $0.8n^2$ edges, are in $E(S, \bar{S})$. □

Lemma 6.5.14 *If $0.1 \cdot n^2 \leq |L_4 \cap S| \leq 2.9 \cdot n^2$, then S expands.*

Proof. As in the second case of the lemma above, the expander edges on L_4 guarantee

$$E(S, \bar{S})| \geq 0.03 \cdot h_0 \cdot n^2 \qquad \text{(by Observation 6.5.1)}$$

as required. □

The last case is when the size $L_4 \cap S$ is small. In this case, we need to consider the sizes of $R_4 \cap S$.

Lemma 6.5.15 *If $|L_4 \cap S| < 0.1 \cdot n^2$ and $|R_4 \cap S| > 2.1 \cdot n^2$, then S expands.*

Proof. This is symmetric to the final case of Lemma 6.5.13. Of the n^2 edges in $L_4 \times R_4$, at most $0.1 \cdot n^2$ are in $(L_4 \cap \bar{S}) \times (R_4 \cap \bar{S})$ (since $L_4 \cap S$ is small), at most $0.1 \cdot n^2$ are in $(L_4 \cap S) \times (R_4 \cap S)$ (since $R_4 \cap S$ is large), and the remaining edges, at least $0.8 \cdot n^2$ edges, are in $E(S, \bar{S})$. □

Lemma 6.5.16 *If $|L_4 \cap S| < 0.1 \cdot n^2$ and $0.1 \cdot n^2 \leq |R_4 \cap S| \leq 2.1 \cdot n^2$, then S expands.*

Proof. The expander edges on R_4 guarantee

$$E(S, \bar{S})| \geq 0.03 \cdot h_0 \cdot n^2 \quad \text{(by Observation 6.5.1)}$$

as required. □

Finally, the intersection of S with both L_4 and R_4 is low. We analyze the subgraphs on $L = L_1 \cup L_2 \cup L_3 \cup L_4$ and on $R = R_1 \cup R_2 \cup R_3 \cup R_4$ separately. We will show that there is a constant h_2 such that

$$|E(L \cap S, L \cap \bar{S})| \geq h_2 \cdot |L \cap S|$$

and

$$|E(R \cap S, R \cap \bar{S})| \geq h_2 \cdot |R \cap S|,$$

which implies

$$\begin{aligned} |E(S, \bar{S})| &\geq |E(L \cap S, L \cap \bar{S})| + |E(R \cap S, R \cap \bar{S})| \\ &\geq h_2 \cdot |L \cap S| + h_2 \cdot |R \cap S| \quad \text{(by assumption)} \\ &= h_2 \cdot |S| \end{aligned}$$

as desired.

Since the cases are symmetric, we focus on L and the proof for R is analogous. We say *S expands within L* if the constant h_2 as above exists. We first deal with the case where $L_4 \cap S$ is at least linear.

Lemma 6.5.17 *If $0.1 \cdot n \leq |L_4 \cap S| < 0.1 \cdot n^2$, then S expands within L.*

Proof. We consider how $L_3 \cap S$ relates to $L_4 \cap S$. First, consider the case when $|L_3 \cap S| \geq 2 \cdot |L_4 \cap S|$. Then we have

$$|E(L \cap S, L \cap \bar{S})| \geq |L_3 \cap S| - |L_4 \cap S| \quad \text{(by Observation 6.5.2)}$$

and we bound this difference from below several times. Since $|L_3 \cap S| \geq 2 \cdot |L_4 \cap S|$, we also have

$$|L_3 \cap S| - |L_4 \cap S| \geq |L_4 \cap S| \quad \text{and} \quad |L_3 \cap S| - |L_4 \cap S| \geq 0.5 \cdot |L_3 \cap S|.$$

In addition,

$$|L_4 \cap S| \geq 0.1 \cdot n \geq 0.05 \cdot |L_1 \cup L_2|. \quad \text{(by assumption)}$$

Hence,

$$\begin{aligned} |E(L \cap S, L \cap \bar{S})| &\geq (0.5 \cdot |L_3 \cap S| + |L_4 \cap S| + 0.05 \cdot |L_1 \cup L_2|)/3 \\ &\geq 0.01 \cdot |L \cap S| \end{aligned}$$

and we are done. The other case is when $|L_3 \cap S| \leq 2 \cdot |L_4 \cap S|$. Then, $|(L_3 \cup L_4) \cap S| \leq |(L_3 \cup L_4)|/2$, and we have

$$|E(L \cap S, L \cap \bar{S})| \geq h_1 \cdot |(L_3 \cup L_4) \cap S|. \quad \text{(by Lemma 6.5.8)}$$

We also have

$$|(L_3 \cup L_4) \cap S| \geq |L_4 \cap S| \geq 0.1 \cdot n \geq 0.05 \cdot |L_1 \cup L_2|.$$

Hence, $|E(L \cap S, L \cap \bar{S})| \geq 0.02 \cdot h_1 \cdot |L \cap S|$ and we are done. □

Lemma 6.5.18 *If $|L_4 \cap S| < 0.1 \cdot n$ and $|L_3 \cap S| \geq 0.2 \cdot n$, then S expands within L.*

Proof. We have that

$$|E(L \cap S, L \cap \bar{S})| \geq |L_3 \cap S| - |L_4 \cap S| \geq |L_3 \cap S|/2. \qquad \text{(by Observation 6.5.2)}$$

Since

$$|L_3 \cap S|/2 > |L_4 \cap S|/4 \quad \text{and} \quad |L_3 \cap S|/2 \geq 0.1 \cdot n > 0.05 \cdot |L_1 \cup L_2|,$$

this gives

$$|E(L \cap S, L \cap \bar{S})| \geq 0.01 \cdot |L_3 \cap S|.$$ □

Lemma 6.5.19 *If $|L_4 \cap S| < 0.1 \cdot n$ and $|L_3 \cap S| < 0.2 \cdot n$, then S expands within L.*

Proof. In this case, we have

$$|E(L \cap S, L \cap \bar{S})| \geq h_1 \cdot |(L_3 \cup L_4) \cap S|. \qquad \text{(by Lemma 6.5.8)}$$

We treat $(L_1 \cap L_2) \cap S$ separately, considering three cases by the sizes of $L_2 \cap S$ and $L_1 \cap S$. The first case is when $|L_2 \cap S| \geq 0.3 \cdot n$. Note that each node in L_2 is connected by an edge to exactly one node in L_3, and these nodes are distinct. So there are at least $0.3n$ edges in $(L_2 \cap S) \times L_3$, of which at most $0.2n$ have their L_3 endpoint in $|L_3 \cap S|$, hence

$$|E(L \cap S, L \cap \bar{S})| \geq 0.1 \cdot n \geq 0.05 \cdot |(L_1 \cup L_2) \cap S|.$$

In the second case, $|L_2 \cap S| < 0.3 \cdot n$ and $|L_1 \cap S| < 0.7 \cdot n$. Then we have

$$|E(L \cap S, L \cap \bar{S})| \geq h_1 \cdot |(L_1 \cup L_2) \cap S|. \qquad \text{(by Lemma 6.5.8)}$$

The final case is when $|L_2 \cap S| < 0.3 \cdot n$ and $|L_1 \cap S| \geq 0.7 \cdot n$. Then

$$\begin{aligned} |E(L \cap S, L \cap \bar{S})| &\geq |L_1 \cap S| - |L_2 \cap S| && \text{(by Observation 6.5.2)} \\ &> 0.4 \cdot n \\ &\geq 0.2 \cdot |L_1 \cup L_2| \\ &\geq 0.4 \cdot |(L_1 \cup L_2) \cap S| \end{aligned}$$

which completes the proof. □

6.5.5. Power-law Graph

As in the case of maximum matching, we first make our reduction graph robust to degree changes. The following reduction is close in spirit to the expander graph reduction. The graph is as follows:

- A $(\log n)$-depth binary forest with a single tree, labelled $L_1 \cup L_2$ as before, with s as the root of the tree.
- A $(\log n)$-depth binary forest with $3n$ trees, labelled $L_3 \cup L_4$. We split this binary forest into sets of n trees each, and label them LU, LM, LL (upper, middle, and lower). The roots of each of the n trees are marked as $LX[i]$, for $1 \leq i \leq n$, $X \in \{U, M, L\}$, and the leaves of the tree with root $LX[i]$ are marked $LX[i, j]$, for $1 \leq j \leq n$.
- A copy of the above structure, with node sets marked R_i instead of L_i, respectively. The root of the single tree of R_1 is the target node t.
- For the matrix M, add the edges $(LM[i, j], RM[j, i])$ and $(LL[i, j], RL[j, i])$ if $M_{ij} = 1$, and add the edges $(LM[i, j], RL[j, i])$ and $(LL[i, j], RM[j, i])$ otherwise.
- Given an input vector u, for each $i \in [n]$, add the edge $(L_2[i], LM[i])$ if $u_i = 1$, and $(L_2[i], LU[i])$ otherwise.
- Given an input vector v, for each $j \in [n]$, add the edge $(R_2[j], RM[j])$ if $v_j = 1$, and $(R_2[j], RU[j])$ otherwise.

Table 6.9.: Degree distribution of the nodes on the left side of the (s, t)-distance reduction graph.

Layer	Deg 1	Deg 2	Deg 3
L_1	0	1	$n-2$
L_2	0	n	0
L_3	0	$2n$	$3n^2 - 5n$
L_4	n^2	$2n^2$	0

Note that this fixes the degree distribution present in the graph regardless of the input (u, M, v), unlike in the case of maximum matching where we needed to compensate for the degrees elsewhere in the graph. Table 6.9 shows the degree distribution on the left side of the reduction graph, regardless of the input bits.

Thus the degree distribution in the entire reduction graph is as follows: $(1, 2n^2), (2, 4n^2 + 6n + 2), (3, 6n^2 - 8n - 4)$. As earlier, choose N such that

$$N > \zeta(\beta) \cdot \max\{(2N_1 + 2n), (2N_2 + 2n) \cdot 2^{\beta}, (2N_3 + 2n) \cdot 3^{\beta}\},$$

and pick any power-law graph G on N nodes. We reduce the count of degree 1, 2 and 3 nodes in the graph as earlier, and embed our reduction graph using these nodes. Unlike in the case of matchings, note that the (s, t)-distance is not affected by the rest of the graph, and thus $uMv = 1$ if and only if $\text{dist}(s, t) \leq 4 \log n + 3$.

6.6. Lower Bounds for Densest Subgraph

Definition 6.6.1 (Vector and Matrix Gadgets) *A* vector gadget *is a 6-edge connected $2d$-regular graph on n nodes. A* matrix gadget *is a 6-edge connected $2d$-regular graph on n^2 nodes with one edge removed.*

Given a constant $d \geq 3$, we work with $(2d)$-regular graphs of two different sizes: an N-node gadget for each vector entry, and an N^2-node gadget for each matrix entry. A 6-edge connected graph is a graph that does not disconnect after the removal of any 5 edges. Such a $(2d)$-regular graph is, e.g., a d-dimensional torus, or a union of d edge-disjoint cycles, each going through all the nodes. Note that the density of a vector gadget is d, and the density of a matrix gadget is $d - \frac{1}{n^2}$. We prove the following theorems.

Theorem 6.6.1 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining an exact densest subgraph, on all N-node graphs with maximum degree $\Delta \leq 7$, with amortized $O(N^{1/4-\varepsilon})$ update time and $O(N^{1/2-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

Theorem 6.6.2 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining an exact densest subgraph, on all N-node graphs with constant degree and constant expansion, with amortized $O(N^{1/4-\varepsilon})$ update time and $O(N^{1/2-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

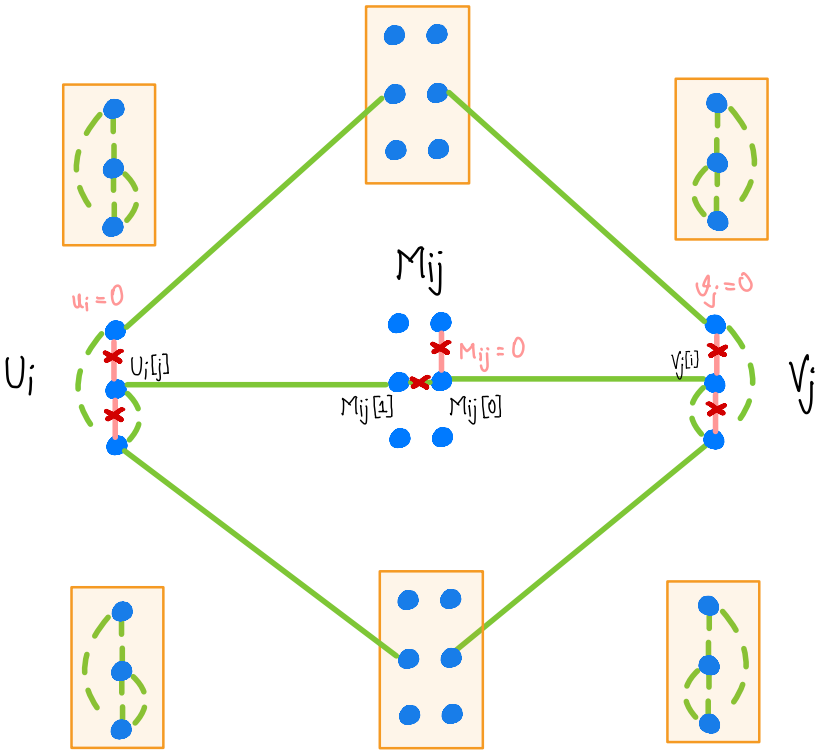


Figure 6.6.: Reduction graph for the densest subgraph lower bound. The light green edges denote the static graph edges, and the pink edges denote the input-dependent edges, when the input is 0.

6.6.1. Constant-Degree Graph

Static Graph. The reduction graph is composed of $2n$ vector gadgets, and n^2 matrix gadgets as follows.

- $2n$ vector gadgets labelled U_i resp. V_i, for $1 \leq i \leq n$. The nodes in each gadget are labelled $U_i[j]$ resp. $V_i[j]$, for $1 \leq j \leq n$.
- n^2 matrix gadgets labelled M_{ij}, for $1 \leq i, j \leq n$. The missing edge in the gadget M_{ij} is between the two nodes labelled $M_{ij}[0]$ and $M_{ij}[1]$.
- An edge from $U_i[j]$ to $M_{ij}[0]$, and an edge from $V_j[i]$ to $M_{ij}[1]$ for all $1 \leq i, j \leq n$. Note that this implies that every node of M_{ij} has degree $2d$ and that the total number of edges incident to at least one node of M_{ij} is $dn^2 + 1$.

The total number of nodes in the reduction graph is therefore

$$N = n^4 + 2n^2 = \Theta(n^4).$$

Input-Dependent Edges.

- If $M_{ij} = 0$, delete an arbitrary edge from the matrix gadget M_{ij}.
- If $u_i = 0$, delete two arbitrary edges from the vector gadget U_i.
- If $v_i = 0$, delete two arbitrary edges from the vector gadget V_i.

Densest Subgraphs in the Graph. We use the following simple lemma in our density proof.

Lemma 6.6.3 *[93] For all numbers a, b, c, d, and r, we have:*

1. *If $\frac{a}{b} \geq r$ and $\frac{c}{d} \geq r$, then $\frac{a+c}{b+d} \geq r$.*
2. *If $\frac{a}{b} \geq r$ and $\frac{c}{d} \leq r$, then $\frac{a-c}{b-d} \geq r$.*

Let ρ be the density of the densest subgraph in the current graph.

Lemma 6.6.4 *If $uMv = 1$, then $\rho \geq d + \frac{1}{n^2+2n}$, and otherwise $\rho < d + \frac{1}{n^2+2n}$.*

Proof. ($\Longrightarrow$) First assume that $uMv = 1$. Then there are indices i, j such that $u_i = M_{ij} = v_j = 1$. Consider the subgraph $S = U_i \cup M_{ij} \cup V_j$. It consists of $n^2 + 2n$ nodes and $d(n^2 + 2n) + 1$ edges. Thus $\rho(S) = d + \frac{1}{n^2+2n}$.

($\Longleftarrow$) Now assume that $uMv = 0$, and that there exists some subset $S \subset V$ with $\rho(S) \geq d + \frac{1}{n^2+2n}$. We first claim that we can modify S in a particular manner without loss of generality, and then derive a contradiction. Specifically, first we remove from S all subgraphs M_{ij} that are not completely contained in S.

Claim 6.6.5 *Let T be a subgraph of a matrix gadget M_{ij} that is not the whole gadget. Then, after removing T from S, it still holds that $\rho(S) \geq d + \frac{1}{n^2+2n}$.*

Proof. Recall that every node in M_{ij} has degree at most $2d$. Let $q = |T| < n^2$. It follows that there are at most qd edges incident to a node of T in $G[S]$: If neither $M_{ij}[0]$ nor $M_{ij}[1]$ belong to T then there are most $qd - 1$ edges incident to at least on node of T in $G[S]$ (all being edges between nodes of T), as at least one node of T must have an edge to a node to the rest of M_{ij} that does not belong to S. If either $M_{ij}[0]$ or $M_{ij}[1]$, but not both belong to T then there are at most $qd - 1$ edges incident to two nodes of T and there is one edge between T and the rest of S. If both $M_{ij}[0]$ and $M_{ij}[1]$ belong to T then there are at most $qd - 2$ edges incident to two nodes of T (as there is no edge between $M_{ij}[0]$ and $M_{ij}[1]$) and there are two edges between T and the rest of S.

Thus the removal of T from S removes at most qd edges and q nodes and, hence by Part 2 of Lemma 6.6.3, $S \setminus T$ has density $\geq d + \frac{1}{n^2+2n}$. □

Next we remove all subgraphs M_{ij} such that $M_{ij} = 0$.

Claim 6.6.6 *For all i, j, if the bit $M_{ij} = 0$ and M_{ij} is fully contained in S, then after removing the corresponding subgraph from S, we still have $\rho(S) \geq d + \frac{1}{n^2+2n}$.*

Proof. As $M_{ij} = 0$, there are dn^2 edges incident to at least one node of M_{ij}. Thus, removing M_{ij} from S removes n^2 nodes and at most dn^2 edges from $G[S]$, i.e., a subgraph of density at most d. By Part 2 of Lemma 6.6.3, $S \setminus M_{ij}$ has density $\geq d + \frac{1}{n^2+2n}$. □

Finally we add to S every vector subgraph that is partially contained in S.

Claim 6.6.7 *For all i, j, after we add any partially contained subgraph U_i or V_j to S it still holds that $\rho(S) \geq d + \frac{1}{n^2+2n}$.*

Proof. We only prove our claim for the gadget U_i, since the proof for the gadget V_j is analogous. Suppose some subset $U \subset U_i$ is contained in S with $|U| = q < n$. We will show that adding $A := U_i \setminus U$ to S adds at least $d(n - q) + 1$ edges. As $\frac{d(n-q)+1}{n-q} > d + \frac{1}{n^2+2n}$, the claim follows from Part 1 of Lemma 6.6.3.

We are left with showing that adding A to S adds at least $d(n-q)+1$ edges. Recall that U_i is a $2d$-regular 6-edge connected graph with n nodes from which 2 edges where removed if $u_i = 0$ and no edges where removed if $u_i = 1$. We consider the following cases:

Case 1: $u_i = 1$ or no removed edge is incident to a node in A. In this case every node in A has degree $2d$ in $G[U_i]$. Let a be the number of edges between U and A, and note that $a \geq 6$ since U_i is 6-edge connected. Thus, the sum of the degrees of the nodes in A is $G[A] = 2d(n-q) - a$. It follows that there are $d(n-q) - a/2$ edges between nodes of A. Thus, the total number of edges incident to nodes of A in $G[U_i]$ is $d(n-q) + a/2 \geq d(n-q) + 3$ and this is a lower bound of the number of edges that are added to S when A is added.

Case 2: $u_i = 0$ and b removed edges belong to (A,U) and c removed edges have both endpoints in A with $b+c \geq 1$. As only 2 edges are removed, $b+c \leq 2$. As U_i is 6-edge connected, let $a \geq 6-b$ be the number of edges between U and A. In this case the sum of the degrees of the nodes of A in $G[A]$ is $2d(n-q)-(a+b)-2c$. Thus, the total number of edges incident to nodes of A in $G[A]$ is $d(n-q)-(a+b)/2-c$. Hence, the total number of edges incident to nodes of A in $G[U_i]$ is $d(n-q)-(a+b)/2-c+a = d(n-q)+(a-b)/2-c$. As $a \geq 6-b$ this is at least $d(n-q)+3-b-c$. Since $b+c \leq 2$, it follows that at least $d(n-q)+1$ edges are added to S when A is added. □

Thus S has the following structure: It has some matrix gadgets of set bits M_{ij} with all the nodes and both outgoing edges present, and it has some vector gadgets of set or unset bits with all nodes present as well. Let x denote the number of matrix gadgets of set bits contained in S, y denote the number of vector gadgets of set bits in S, and z denote the number of vector gadgets of unset bits in S. We now consider two cases:

Case 1: $y+z=0$. Then S consists only of x matrix gadgets and no vector gadgets. Thus $\rho(S) = \frac{x\cdot(dn^2-1)}{n^2x} < d$, which is a contradiction

Case 2: $y+z>0$. Then the density of S is given by

$$\rho(S) = \frac{x\cdot(dn^2+1) + y\cdot dn + z\cdot(dn-2)}{n^2x + ny + nz}$$

We claimed that $\rho(S) \geq d + \frac{1}{n^2+2n}$, which is the same

$$2nx \geq ny + (2n^2+5n)\cdot z$$

Since $uMv = 0$, for every matrix gadget M_{ij} of a set bit, either the corresponding U_i or V_j must be unset. Thus we assign at most n matrix gadgets to each unset vector gadget, giving us that $x \leq nz$, giving

$$2n^2z \geq 2nx \geq ny + (2n^2+5n)\cdot z,$$

which is a contradiction as desired since $y+z>0$. □

Complexity of the Reduction. We now prove the theorem.

Theorem 6.6.1 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining an exact densest subgraph, on all N-node graphs with maximum degree $\Delta \leq 7$, with amortized $O(N^{1/4-\varepsilon})$ update time and $O(N^{1/2-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

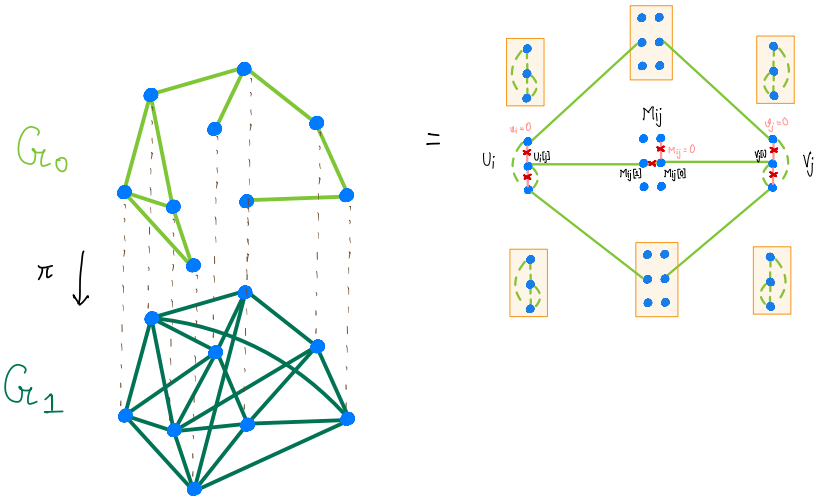


Figure 6.7.: Reduction graph for the expander densest subgraph lower bound. It is similar to Figure 6.6, except the original reduction graph from the constant-degree reduction is represented in light green and the expander is denoted in dark green.

Proof. Consider the reduction graph above with $d = 3$. It consists of $N = n^4 + 2n^2 = \Theta(n^4)$ nodes. Every time we get a new (u, v) input vector pair, we reinsert all the removed edges in the vector gadgets U_i, V_i, for all $1 \leq i \leq n$, and then delete edges according to the new input vectors. This takes $O(n)$ updates in total. After that, we query once for the density of the densest subgraph in this new graph, and return 1 if and only if $\rho \geq 3 + \frac{1}{n^2+2n}$.

Thus for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ query. In total, checking n pairs takes us $O(n^2)$ updates and $O(n)$ query. If there were an algorithm for maintaining the density of the densest subgraph on constant-degree graphs with update time $O(N^{1/4-\varepsilon})$ (i.e., $O(n^{1-4\varepsilon})$) and query time $O(N^{1/2-\varepsilon})$ (i.e., $O(n^{2-4\varepsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-4\varepsilon})$ time, contradicting the OMv conjecture. □

6.6.2. Expander Graph

The extension required to make the reduction hold even for expanders for the densest subgraph problem is simpler than the corresponding extensions for the previous problems. Pick d such that there is a d'-regular expander for some $d' \leq d - 2$. The static graph is constructed as follows.

- The reduction graph G_0 from Section 6.6.1 as a subgraph together with the corresponding input-dependent edges.
- A (d')-regular expander graph G_1 on n^4+2n^2 nodes, for some $d' \leq d-2$, with constant expansion h_0, together with a bijection π from the nodes of G_0 to the G_1.
- An edge between every node v of G_0 and the node $\pi(v)$ of G_1, giving a perfect matching connecting the nodes of G_0 with these of G_1.

The total number of nodes in the reduction graph is $N = 2n^4+4n^2 = \Theta(n^4)$.

The density arguments in this reduction graph are similar to the one in the constant-degree case, as in Lemma 6.6.4. The proof follows from the following fact: for any subset T with density $> d$, removing all of the nodes of G_1 from T cannot decrease the density of T, since each node in G_1 has degree at most $d-1$ in $G[T]$. Thus, in the proof of Lemma 6.6.4 in the setting where $uMv = 0$, we add a first step removing all the nodes of G_1 from the subset S and then proceed as before. This leads to the same lower bounds as in the constant-degree case. We have the following theorem for expander graphs:

Theorem 6.6.2 *Given any constant $\varepsilon > 0$, there is no dynamic algorithm maintaining an exact densest subgraph, on all N-node graphs with constant degree and constant expansion, with amortized $O(N^{1/4-\varepsilon})$ update time and $O(N^{1/2-\varepsilon})$ query time, unless the* OMv *conjecture is false.*

To verify that the graph is indeed an h_1-expander graph for some constant $h_1 > 0$, we follow a similar pattern as in the proof of expansion for maximum matching in Section 6.4.3. Consider a reduction graph with $V = U \cup U'$, where U' is the node set of the expander G_1. Let S be some set of nodes (we do not bound $|S|$). We have the following observations.

Observation 6.6.1 *Let $0 < c < 1/2$, that may depend on n. If $c \cdot |U'| \leq |S \cap U'| \leq (1-c) \cdot |U'|$, then $E(S, \bar{S})| \geq c \cdot h_0 \cdot |U'|$.*

Proof. Consider first the case $|S \cap U'| \leq |U'|/2$. We have

$$\begin{aligned} |E(S,\bar{S})| &\geq |E(S,\bar{S}) \cap U'| \\ &\geq h_0 \cdot |S| && \text{(by expansion of } U') \\ &\geq c \cdot h_0 \cdot |U'|. && \text{(since } |S \cap U'| \leq |U'|/2) \end{aligned}$$

Otherwise, apply the analogous argument on the set $U' \setminus S$. □

Observation 6.6.2 $|E(S, \bar{S})| \geq |U \cap S| - |U' \cap S|$.

Proof. Every node in $U \cap S$ has a matching partner (by the bijection used to map the nodes of U to U') in U'. In total, each of these nodes has $|U \cap S|$ edges going to their matching partners, of which at most $|U' \cap S|$ partners are in S. Hence, $|E(S, \bar{S})| \geq |U \cap S| - |U' \cap S|$. □

Observation 6.6.3 *Let $0 < c < 1/2$ constant. If $|U \cap S| \leq c \cdot |U' \cap S|$ then $|E(S, \bar{S})| \geq (1-c) \cdot |U' \cap S|$.*

Proof. If $|U \cap S| \leq c \cdot |U' \cap S|$, consider the matching partners of the nodes in $U' \cap S$: there are at least $|U' \cap S|$ such partners in U, of which at most $c \cdot |U \cap S|$ are in S, and the others are in $U \cap \bar{S}$. Hence $|E(U \cap \bar{S}, U' \cap S)| \geq (1-c) \cdot |U' \cap S|$. □

Lemma 6.6.8 *The reduction graph is an expander.*

Fix a set $S \subseteq V$ of nodes, with $|S| \leq n^4 + 2n^2 = (|U \cup U'|)/2$. We consider different ranges of $|U' \cap S|$, and show that for each of them, $|E(S, \bar{S})| \geq h_1 \cdot |S|$ for some constant h_1.

Lemma 6.6.9 *If $|U' \cap S| > 0.9 \cdot |U'|$, then S expands.*

Proof. The inequalities $|U| \leq |U'|$ and $|S| \leq |U \cup U'|/2$ imply $|U \cap S| \leq 0.1 \cdot |U'|$, and hence

$$|U' \cap S| > (1/9) \cdot |U \cap S| > 0.1 \cdot |U \cap S|.$$

We have

$$\begin{aligned}
&\text{(by Observation 6.6.3)} && |E(S,\bar{S})| \geq 0.9 \cdot |U' \cap S| \\
&\text{(by assumption)} && \geq 0.81 \cdot |U'| \\
&\text{(since } |U| \leq |U'|\text{)} && > 0.4 \cdot |U \cup U'| \\
&\text{(since } |S| \leq |U \cup U'|/2\text{)} && \geq 0.8 \cdot |S|
\end{aligned}$$

as required. □

Lemma 6.6.10 *If* $0.1 \cdot |U'| \leq |U' \cap S| \leq 0.9 \cdot |U'|$*, then* S *expands.*

Proof. Since $|U| \leq |U'|$ and $|S| \leq |U \cup U'|/2$, we have

$$\begin{aligned}
&\text{(by Observation 6.6.1)} && |E(S,\bar{S})| \geq 0.1 \cdot h_0 \cdot |U'| \\
&\text{(since } |U| \leq |U'|\text{)} && \geq 0.05 \cdot h_0 \cdot |U \cup U'| \\
&\text{(since } |S| \leq |U \cup U'|/2\text{)} && \geq 0.1 \cdot h_0 \cdot |S|
\end{aligned}$$

as required. □

Lemma 6.6.11 *If* $|U' \cap S| < 0.1 \cdot |U'|$*, then* S *expands.*

Proof. First, consider the setting where $|U \cap S| > 2 \cdot |U' \cap S|$. In this case, add $0.5 \cdot |U \cap S| - 1.5 \cdot |U' \cap S|$ to both sides of the inequality and multiply by $2/3$ to get

$$|U \cap S| - |U' \cap S| > 1/3 \cdot (|U \cap S| + |U' \cap S|) = |S|/3.$$

We have

$$\begin{aligned}
&\text{(by Observation 6.6.2)} && |E(S,\bar{S})| \geq |U \cap S| - |U' \cap S| \\
& && > |S|/3
\end{aligned}$$

Else, we are in the case where $|U \cap S| \leq 2 \cdot |U' \cap S|$. In this case, we have that

$$|S| = |U \cap S| + |U' \cap S| \leq 3 \cdot |U' \cap S|.$$

Thus,

$$\begin{aligned}
&\text{(by the expansion of } U'\text{)} && |E(S,\bar{S})| \geq h_0 \cdot |U' \cap S| \\
& && \geq (h_0/3) \cdot |S|
\end{aligned}$$

as required. □

6.6.3. Power-law Graph

Let $d = 3$. We show our densest subgraph lower bounds for $\beta > 2.74$. We use a reduction graph similar to the one in the constant-degree case for our reduction. Our reduction graph consists of the vector and matrix gadgets as before, along with new nodes we introduce to moderate the degree. We add a cycle C_i^u on four new nodes for each vector gadget, and a cycle C_{ij} on four new nodes for each matrix gadget.

- $2n$ vector gadgets labelled U_i resp. V_i, for $1 \leq i \leq n$. The nodes in each gadget are labelled $U_i[j]$ resp. $V_i[j]$, for $1 \leq j \leq n$.
- $2n$ cycles C_i^u and C_j^v on 4 nodes each, with the nodes labelled $C_i^w[x]$ for $w \in \{u, v\}$, $x \in \{a, b, c, d\}$

- n^2 matrix gadgets labelled M_{ij}, for $1 \le i, j \le n$. The missing edge in the gadget M_{ij} is between the two nodes labelled $M_{ij}[0]$ and $M_{ij}[1]$.
- n^2 cycles C_{ij} on four nodes $C_{ij}[x]$ for $x \in \{a, b, c, d\}$ each.
- An edge from $U_i[j]$ to $M_{ij}[0]$, and an edge from $V_j[i]$ to $M_{ij}[1]$ for all $1 \le i, j \le n$.
- If $M_{ij} = 0$, remove two arbitrary edges, say the edges $(M_{ij}[a], M_{ij}[b])$ and $(M_{ij}[c], M_{ij}[d])$, and the two edges $(C_{ij}[a], C_{ij}[b])$ and $(C_{ij}[c], C_{ij}[d])$, and add the edges $(M_{ij}[x], C_{ij}[x])$ for $x \in \{a, b, c, d\}$ to the graph.
- Given an input vector u, for each $i \in [n]$, do the following if $u_i = 0$. Remove two arbitrary edges from the vector gadget U_i, say $(U_i[a], U_i[b])$ and $(U_i[c], U_i[d])$, and also remove the two edges $(C_i^u[a], C_i^u[b])$ and $(C_i^u[c], C_i^u[d])$. Add the edges $(U_i[x], C_i^u[x])$ for $x \in \{a, b, c, d\}$. Similarly for an input vector v.

Note that each node in a vector gadget always has degree $2d + 1$, and each node in a matrix gadget has degree $2d$, regardless of input. Further, the degrees of the nodes in all the cycles are exactly 2 for all inputs. The degree distribution of the reduction graph is $(2, 4n^2 + 8n), (2d, n^4), (2d + 1, 2n^2)$. Choose N such that

$$N > \zeta(\beta) \cdot \max\{(4n^2 + 8n) \cdot 2^\beta, (n^4) \cdot (2d)^\beta, (2n^2) \cdot (2d + 1)^\beta\}$$

In a power-law graph with N nodes, the sum of degrees of all nodes is given by $\zeta(\beta - 1) \cdot N/\zeta(\beta)$, while the number of nodes of degree 1 is $N/\zeta(\beta)$. Thus for all β such that $\zeta(\beta - 1) < 2$, more than half the total degree comes from nodes of degree 1, since

$$N_1' = \frac{N}{\zeta(\beta)} > \frac{\zeta(\beta - 1)}{2} \cdot \frac{N}{\zeta(\beta)} = \frac{\sum_{v \in V} \deg(v)}{2} \tag{6.1}$$

This is true in particular for all $\beta > 2.74$. Thus now we first construct the reduction graph as above on $n^4 + 6n^2 + 8n$ nodes, and this does not exceed N_d' for any d because of the definition of N. Now for every other degree that needs to be satisfied of degree $d' > 1$, we simply add a new node and create a star with d' nodes of degree 1 attached to it. Note that we can do this for all the remaining nodes because of Equation (6.1). If there are any remaining degree 1 requirements to be satisfied, we simply add a perfect matching on that many nodes.

None of the stars or the perfect matchings can be part of a subgraph of density $> d$, since they all have density < 1. Thus any subgraph of density $> d$ must be from our reduction graph. Further, note that the cycle gadgets can be removed from any subgraph of density $> d$, since they have degree exactly 2. Thus $\rho \ge d + \frac{1}{n^2+2n}$ if and only if $uMv = 1$, which proves our claim.

6.7. Partially Dynamic Lower Bounds

6.7.1. Dynamic (s, t)-Distance

We show that our dynamic (s, t)-distance lower bound for constant-degree graphs also holds for partially dynamic algorithms.

Static Graph.

- A $(\log n)$-depth binary forest with a single tree. The internal nodes are marked L_1 and the leaves L_2; the root of the tree is the source node s, and the nodes of L_2 are marked as $L_2[i]$ for $1 \leq i \leq n$.
- A $(\log n)$-depth binary forest with n trees. The internal nodes are marked L_3 and the leaves L_4. The roots of each of the n trees are marked as $L_3[i]$, for $1 \leq i \leq n$, and the leaves of the tree with root $L_3[i]$ are marked $L_4[i, j]$, for $1 \leq j \leq n$.
- An edge from $L_2[i]$ to $L_3[i]$ for $1 \leq i \leq n$.
- n paths $P[i]$ on n nodes. The nodes of the path $P[i]$ are marked $P[i, n + 1 - j]$, $1 \leq j \leq n$. Edges from $L_4[i, j]$ to $P[i, j]$ for all $1 \leq i, j \leq n$.
- A $(\log n)$-depth binary forest with n trees. The internal nodes are marked L_5 and the leaves L_6. The roots of each of the n trees are marked as $L_5[i]$, for $1 \leq i \leq n$, and the leaves of the tree with root $L_5[i]$ are marked $L_6[i, j]$, for $1 \leq j \leq n$.
- An edge from $P[i, 1]$ to $L_5[i]$ for all $1 \leq i \leq n$.
- A copy of the above structure, with node sets marked R_i and $Q[i]$ instead of L_i and $P[i]$. The root of the single tree of R_1 is the target node t.

Input-Dependent Edges. We add the following edges to the initial graph based on the matrix.

- If $M_{ij} = 1$, add the edge $(L_6[i, j], R_6[j, i])$.

We perform the following deletions for the j^{th} input vectors (u^j, v^j).

- If $u_i^j = 0$, delete the edge $(L_4[i, j], P[i, j])$.
- If $v_i^j = 0$, delete the edge $(R_4[i, j], Q[i, j])$.

Before the $(j + 1)^{th}$ input vector arrives, delete the edges $(L_4[i, j], P[i, j])$ and $(R_4[i, j], Q[i, j])$ for all $1 \leq i \leq n$. It is easy to see that there is a path of length $6 \log n + 5 + 2j$ if and only if $u^j M v^j = 1$.

Since the reduction graph consists of $\Theta(n^2)$ nodes and we make $O(n)$ updates and 1 query for each input pair, we get the same lower bounds as in the fully-dynamic setting. Note that this lower bound can be made to work for the insertions only setting as well by reversing the path P and Q.

6.7.2. Dynamic Maximum Matching

We use the following reduction graph to make our fully dynamic maximum matching lower bounds hold for the partially dynamic setting as well.

Static Graph.

- A reduction gadget with n subgadgets of size $2n + 2$, on a set $L_1 \cup L_2$. The subgadgets are labelled $LE[j]$ for $1 \leq j \leq n$, and the nodes of subgadget $LE[j]$ are labelled $L_1[j, i]$ or $L_2[j, i]$ for $0 \leq i \leq n$. The path in each subgadget goes from $L_1[j, 0]$ to $L_2[j, n]$.
- A reduction gadget with n subgadgets of size $2n + 2$, on a set $L_3 \cup L_4$. The subgadgets are labelled $LF[i]$ for $1 \leq i \leq n$, and the nodes of subgadget $LF[i]$ are labelled $L_3[i, j]$ or $L_4[i, j]$ for $0 \leq j \leq n$. The path in each subgadget goes from $L_3[i, 0]$ to $L_3[i, n]$.
- Edges from $L_2[j, i]$ to $L_3[i, j]$ for all $1 \leq i, j \leq n$.

- A reduction gadget with n subgadgets of size $2n + 2$, on a set $L_5 \cup L_6$. The subgadgets are labelled $LG[i]$ for $1 \leq i \leq n$, and the nodes of subgadget $LG[i]$ are labelled $L_5[i, j]$ or $L_6[i, j]$ for $0 \leq j \leq n$. The path in each subgadget goes from $L_5[i, 0]$ to $L_6[i, n]$.
- An edge from $L_4[i, n]$ to $L_5[i, 0]$ for all $1 \leq i \leq n$.
- A copy of the above structure, with node sets marked R_i instead of L_i.

Input-Dependent Edges. We add the following edges to the initial graph based on the matrix.

- If $M_{ij} = 1$, add the edge $(L_6[i, j], R_6[j, i])$.

We perform the following deletions for the j^{th} input vectors (u^j, v^j):

- Delete the edge $(L_1[j, 0], L_2[j, 0])$.
- If $u_i^j = 0$, delete the edge $(L_2[j, i], L_3[i, j])$.
- If $v_i^j = 0$, delete the edge $(R_2[j, i], R_3[i, j])$.

Before the $(j + 1)^{th}$ input vector arrives, delete the edges $(L_2[j, 0], L_1[j, 1])$, $(R_2[j, 0], R_1[j, 1])$, $(L_2[j, i], L_3[i, j])$, and $(R_2[j, i], L_3[i, j])$ for all $1 \leq i \leq n$. It is easy to see that there is a matching with only $4j - 2$ nodes unmatched if and only if $u^j M v^j = 1$.

Since the reduction graph consists of $\Theta(n^2)$ nodes and we make $O(n)$ updates and 1 query for each input pair, we get the same lower bounds as in the fully-dynamic setting. This reduction can be made to work for the insertions only setting by reversing the above construction.

Part II.

Differential Privacy

7. Private Histograms

The everyday documents of the Company state—for example, the Tamil palm-leaf records of lower-echelon kanakkuppillai—were rarely ordered systematically, and exist today in uncatalogued manuscript bundles.

Bhavani Raman, *Document Raj*

7.1. Introduction

Maintaining continual sums of a stream of numbers is an integral subroutine in various applications, including iterative first-order methods in machine learning and online convex optimization, which require maintaining the sum of all the past gradients to make future decisions. Private mechanisms for these problems thus rely on privately maintaining continual sums of gradients [116–122], or more generally of a stream of numbers [22, 23, 123–126], with minimal error.

Since the initial work of Dwork et al. [22] and Chan et al. [23], achieving an asymptotic improvement in the additive error of private continual counting or proving that the current bound is optimal has become a major open problem in the field. Recent work has concentrated on non-asymptotic improvements: One line of work [123, 126, 127] improved the constant term in front of the larger asymptotic terms in the additive error. Another line [18] improved the error to optimal on sparse streams in the insertions-only setting with an error bound that is parameterized in the maximum output value. We present two improvements to a high-dimensional generalization of this problem in the spirit of the latter parameterized result of Dwork et al. [18].

At a high level, Dwork et al. [18] obtain their parameterized improvements by partitioning the stream of inputs into *intervals* using SVT, then batching all the inputs within an interval as a single input to a black-box continual counting mechanism. They show that the number of intervals created is proportional to the sparsity of the input stream. This reduces the error dependence on the length of the input stream to just its sparsity.

Extending continual sum of a stream of scalars to maintaining continual coordinate sums of a stream of d-dimensional vectors is a natural generalization when one needs to keep track of high-dimensional information, as is the case in the above machine learning applications. In this chapter, we study the Histogram problem. We also show how to more accurately answer *histogram queries* that do not necessarily require maintaining the entire histogram, for example, the minimum or median column sum.

Motivating Questions. For private continual counting, the best known lower bound on the additive error is $\Omega(\log T)$ [22] against an upper bound of $O(\log^2 T)$ [22, 23]. In the parameterized setting, Dwork et al. [18]'s mechanism has $O(\log^2 n + \log T)$ additive error for insertions-only streams, where n is the largest output of the mechanism on the entire stream. Since this mechanism asymptotically improves on all other mechanisms for streams

[22]: Dwork et al. (2010), "Differential privacy under continual observation"

[23]: Chan et al. (2011), "Private and Continual Release of Statistics"

[116]: Kairouz et al. (2021), "Practical and Private (Deep) Learning Without Sampling or Shuffling"

[117]: Zhang et al. (2022), "Differentially Private Online-to-batch for Smooth Losses"

[118]: Denisov et al. (2022), "Improved Differential Privacy for SGD via Optimal Private Linear Operators on Adaptive Streams"

[119]: Choquette-Choo et al. (2023), "Multi-Epoch Matrix Factorization Mechanisms for Private Machine Learning"

[120]: Asi et al. (2023), "Near-Optimal Algorithms for Private Online Optimization in the Realizable Regime"

[121]: Choquette-Choo et al. (2023), "(Amplified) Banded Matrix Factorization: A unified approach to private training"

[122]: Xu et al. (2023), "Federated Learning of Gboard Language Models with Differential Privacy"

[123]: Fichtenberger et al. (2023), "Constant Matters: Fine-grained Error Bound on Differentially Private Continual Observation"

[124]: Henzinger et al. (2023), "Almost Tight Error Bounds on Differentially Private Continual Counting"

We discuss ε-dp in this section, and drop ε^{-1} and $\log(1/\beta)$ factors for simplicity.

Insertion-only streams have $\mathscr{U} \subseteq [0, \infty)$, while turnstile streams have $\mathscr{U} \subseteq (-\infty, \infty)$.

with continual count $T^{o(1)}$ and is optimal when the count is $O(2^{\sqrt{\log T}})$, we ask the following generalization to d-dimensional streams.

[18]: Dwork et al. (2015), "Pure Differential Privacy for Rectangle Queries via Private Partitions"

[125]: Andersson et al. (2023), "A Smooth Binary Mechanism for Efficient Private Continual Observation"

[126]: Henzinger et al. (2025), "Improved Differentially Private Continual Observation Using Group Algebra"

[127]: Andersson et al. (2025), "Count on Your Elders: Laplace vs Gaussian Noise"

[128]: Jain et al. (2023), "The Price of Differential Privacy under Continual Observation"

> Can we exploit sparsity of outputs for private continual Histogram and histogram queries to obtain a smaller additive error?

The requirement here for histogram queries is stronger than that of Histogram, since the histogram query of, say, the minimum column sum has a much smaller output than that of the entire Histogram.

The best known lower bound here is $\Omega(d+\log T)$ [22, 128], and we show that existing mechanisms for Histogram have an additive error of $\Omega(d \log T)$. Thus, achieving $o(d \log T)$ error necessitates new approaches. Our first mechanism improves on existing mechanisms for streams and queries with maximum output value $T^{o(1)}$, and breaks the $\Omega(d \log T)$ barrier on streams with maximum output value $o(2^{\sqrt{\log T}})$, which answers the highlighted question in the affirmative. The mechanism works by partitioning the input stream similarly to Dwork et al. [18], while taking the interaction between all d columns into account. This extension to the multi-dimensional setting involves overcoming multiple technical difficulties which we detail later.

Another shortcoming of the mechanism of Dwork et al. [18] is that it does not allow negative entries. It implicitly assumes that the continual count is increasing *monotonically*, which might not be true for turnstile streams. This motivates our second main question.

> Can we obtain parameterized improvements (similar to the improvements of Dwork et al. [18]) for turnstile streams?

Improvements in the turnstile model are important since continual histogram mechanisms in this model are used as subroutines in more advanced private continual release mechanisms, such as in the above machine learning applications, fully dynamic graph mechanisms [123, 129], for k-means clustering in Euclidean spaces [130], and also to count the *difference sequence* [129] of an insertions-only stream.

[129]: Fichtenberger et al. (2021), "Differentially Private Algorithms for Graphs Under Continual Observation"

[130]: Tour et al. (2024), "Making Old Things New: A Unified Algorithm for Differentially Private Clustering"

We answer our second question in the affirmative, by parameterizing on the *number of fluctuations*, K, which is the number of times a row vector in the input is different from its immediately preceding row vector. We show that the improvements achieved by our first mechanism also carry over to the second, giving a mechanism for Histogram on turnstile streams that breaks the $\Omega(d \log T)$ barrier on streams with small K. At a high level, while existing mechanisms use SVT to update outputs when the function value changes a lot, our mechanism instead uses SVT to check when the *slope* of the function changes a lot, and we show that one can still obtain improved error bounds under this condition.

We focus on pure dp and present the approximate dp bounds in Section 7.9.

Prior Work. On the lower bounds side, Jain et al. [128] study Histogram as well as maintaining the maximum column sum (MaxSum) and its coordinate (SumSelect) privately and show that the additive error[1] must be $\Omega(d)$. Combined with the counting lower bound of $\Omega(\log T)$ [22], this gives an $\Omega(d + \log T)$ bound. Cohen et al. [131] give a lower bound of $\Omega(\min\{n, \log T\})$ for continual counting. On the upper bound side, Jain et al. [128] give an $O(d \log d \log^3 T)$ bound on the additive error. In the turnstile model, composing d binary tree counting mechanisms of Chan et al. [23], and suitably scaling the privacy parameter and failure probability of

1: We focus only on their bounds that are subpolynomial in T since we consider the setting where $T \gg d$.

[131]: Cohen et al. (2024), "Lower Bounds for Differential Privacy Under Continual Observation and Online Threshold Queries"

each mechanism, gives an error of $O(d \log^2(dT))$. For insertion-only streams, combining the sparse-vector based mechanism of Dwork et al. [18] with observations by Qiu and Yi [132] and Chan et al. [23] gives an error bound of $O(d \log^2(dn_{\max}) + d \log T)$, where $n_{\max}$ is the maximum column sum.

[132]: Qiu et al. (2022), "Differential Privacy on Dynamic Data"

As seen from the bounds above, all existing histogram mechanisms have an $O(d \log T)$ term in their upper bounds. We show that this dependency is inherent, by proving that any histogram mechanism obtained as a composition of d counting mechanisms has $\Omega(d \log T)$ error. This implies that any mechanism that beats this barrier must necessarily take into account the interaction between the different columns.

7.2. Our Results

We present two new mechanisms to answer the questions raised above. Both mechanisms achieve a parameterized error bound which *does not have an additive* $d \log T$, but is instead of the form $O(d \log^2 \rho + \log T)$, with ρ, one of the above-mentioned parameters, potentially much smaller than T.

Mechanism 7.1: Our first mechanism gives new parameterized upper bounds on the additive error in the insertions-only model for computing a large class of histogram-based queries that include Histogram, MaxSum and MinSum[2], SumSelect, as well as Quantile$_p$[3] and TopK[4], where the parameter depends on the queries answered. More specifically, the additive error of the mechanism is $O\left(d \log^2(dq^*) + \log T\right)$, where q^* is the *maximum query value for the given input data at any time step.* Thus if $q^* = o(2^{\sqrt{\log T}})$, our result breaks the $\Omega(d \log T)$ bound and is better than what can be achieved by the previous mechanisms, even when the maximum column sum $n_{\max} = \Omega(T)$. Our mechanism does not need to be given q^* or T at initialization. The lower bound of $\Omega(d + \log T)$ mentioned earlier implies that the dependency on d or $\log T$ cannot be removed.

2: Return the minimum column sum.
3: Return the p-th quantile column sum.
4: Return the k-th largest column sums.

More generally, we define a class of real-valued queries, called *monotone histogram queries*, that subsumes the queries mentioned above. These queries are *monotonically increasing* when a new (nonnegative) row is added, and have low *sensitivity*. Informally, a query's *sensitivity* is the maximum difference between its output values on two neighboring streams.

Definition 7.2.1 (Monotone histogram query) *A function* $q : \{\{0,1\}^d\}^* \to \mathbb{R}_{\geq 0}$ *is a* monotone histogram query *if it is monotonically increasing; has sensitivity* ≤ 1*; and depends only on the input column sums*[5].

5: symmetrically, this also works for monotonically decreasing functions in the deletions-only setting.

Each individual coordinate sum satisfies this definition, as do all the queries described above. Histogram can be obtained with d monotone histogram queries, namely, each coordinate sum.

Theorem 7.2.1 *Let* $x = x^1, \ldots, x^T$ *be an insertions-only stream, and* $q_1, \ldots, q_m$ *be* m *monotone histogram queries. Mechanism 7.1 is* ε*-differentially private, and answers* $(q_1(x^1, \ldots, x^t), \ldots, q_m(x^1, \ldots, x^t))$ *at all time steps* t *with a bound on the* ℓ_∞*-error of*

$$O\left(\left(d \log^2(dmq^*/\beta) + m \log(mq^*/\beta) + \log T\right)\varepsilon^{-1}\right)$$

that holds with probability $\geq 1 - \beta$ *simultaneously over all time steps, where* $q^* = \max_{k \in [m]} q_k(x)$*. Neither* T *nor* q^* *need to be given as input to the*

Table 7.1.: ℓ_∞-error of ε-dp mechanisms for continual histogram queries with constant ε, constant failure probability in the insertions-only model, where $n_{\max}$ is the maximum column sum, q^* is the maximum query output, and K is the number of fluctuations in the input stream. K and q^* do not need to be given to the mechanism. The partitioning follows from Dwork et al. [18], Chan et al. [23] and Qiu and Yi [132].

Mechanism	Histogram	m histogram queries
Jain et al. [128]	$O(d \log d \log^3 T)$	$O(d \log d \log^3 T)$
Binary Tree	$O(d \log^2(dT))$	$O(d \log^2(dT))$
Partitioning	$O(d \log^2(dn_{\max}) + d \log T)$	$O(d \log^2(dn_{\max}) + d \log T)$
Theorem 7.2.1	$O(d \log^2(dn_{\max}) + \log T)$	$O(d \log^2(dq^*) + m \log(mq^*) + \log T)$
Theorem 7.2.2	$O(d \log^2(dK) + \log T)$	$O(d \log^2(dK) + \log T)$

mechanism.

Answering a single monotone histogram query using the best existing ε-dp mechanisms requires computing a full histogram, which gives error $O(d \log^2(dn_{\max}) + d \log T)$. Theorem 7.2.1 improves on these bounds in three significant ways: (1) The $d \log T$ term is replaced by a $\log T$ term, giving the first mechanism that achieves a $o(d \log T)$ bound on sparse outputs. (2) The polylogarithmic dependency of $d \log^2(dn_{\max})$ on the maximum column sum $n_{\max}$ is reduced to a polylogarithmic dependency of $d \log^2(dq^*)$ on the *maximum query value* q^*. For monotone histogram queries, we have $q^* \leq n_{\max}$, and q^* could be much smaller than $n_{\max}$, when computing, say, the minimum column sum or the median column sum. This is true, for example, on streams with a power-law distribution of the column sums. (3) We can answer up to m queries *without incurring an extra additive* $O(m \log T)$ *error*, which would happen when using standard composition. We extend these results to natural-numbered inputs, to (ε, δ)-dp, and to different neighboring definitions in Section 7.6.

Table 7.2.: ℓ_∞-error for ε-dp mechanisms with constant ε and constant failure probability in the turnstile model, where K is the number of fluctuations.

Mechanism	Histogram
Jain et al. [128]	$O(d \log d \log^3 T)$
Binary Tree	$O(d \log^2(dT))$
Theorem 7.2.2	$O(d \log^2(dK) + \log T)$

Mechanism 7.2: Our second mechanism gives new parameterized upper bounds in the turnstile model for continual counting and Histogram. This mechanism has an additive error of $O(d \log^2(dK) + \log T)$, where K is the *number of fluctuations*, which is the number of time steps where $x^t \neq x^{t+1}$. It does not need to be given K or T at initialization. This is the first improvement in the turnstile model since the 2011 bound of $O(\log^2 T)$ for continual counting (and $O(d \log^2(dT) + d \log T)$ for Histogram) by Chan et al. [23]. Recall that the parameterized result of Dwork et al. [18] only works for insertions-only streams.

Our result generalizes the bound of Dwork et al. [18] in two regards: to possibly negative-valued inputs, and to d-dimensional inputs. Our bound improves or matches their additive error on insertions-only streams, since $K \leq 2dn_{\max}$. However, K might be considerably smaller: a data stream that contains $n/2$ ones followed by $n/2$ zeros has $K = 1$, while $n_{\max} = T/2$. Moreover, real world data often show strong time correlations, leading to a small value of K. Examples include recommendation systems, where movies or products that are popular at a given time are likely to be rated consecutively by more people, and outlier monitoring processes, where many of the generated reports are identical (when nothing special has happened).

Theorem 7.2.2 *Let $x = x^1, \ldots, x^T$ be a turnstile stream. Let K be the total number of times $x^t \neq x^{t+1}$, for all $t < T$. Mechanism 7.2 is ε-differentially private, and outputs an estimate of the histogram at all time steps t with ℓ_∞-error bound $O\left(\left(d \log^2(dK/\beta) + \log T\right) \varepsilon^{-1}\right)$ that holds with probability at least $1 - \beta$ simultaneously over all time steps. Neither T nor K need to be given as input to the mechanism.*

Our result also gives insights about stronger continual sum lower bounds:

- Input streams consisting of $O(2^{\sqrt{\log T}})$ non-zero entries will not lead to a stronger lower bound for continual counting [18].
- Even further, the input streams would need to change frequently, i.e., $\omega(2^{\sqrt{\log T}})$ times, unlike the streams used in the current lower bounds of [22, 131].

Note that a tight lower bound in K, i.e., a lower bound of $\Omega(\log^2 K + \log T)$ for $d = 1$ that holds for all values of K would also imply a lower bound of $\Omega(\log^2 T)$ for continual counting (since K could be as large as T), which is a major open problem in the area.

7.3. Differential Privacy Against an Adaptive Adversary

As a subroutine, we use a continual histogram mechanism that works in the stronger *adaptive continual release model* defined by Jain et al. [128]. In this model, the mechanism M interacts with a *randomized adversarial process* Adv that has no restrictions on its time or space complexity. It knows the mechanism M and all its inputs and outputs up to the current time step, but *not* its random coin flips. Based on this, Adv has to choose the input to M for the next time step.

[128]: Jain et al. (2023), "The Price of Differential Privacy under Continual Observation"

Event-level neighboring inputs are modelled as follows. All time steps except one are *regular*, and the adversary is allowed to adaptively determine when the special *challenge* time step occurs. At a regular time step, Adv outputs one value x^t. At a challenge time step, Adv outputs two values $x^t_{(L)}$ and $x^t_{(R)}$ such that $x^1, \dots, x^t_{(L)}, x^{t+1}, \dots$ and $x^1, \dots, x^t_{(R)}, x^{t+1}, \dots$ are neighboring[6]. At the challenge time step, an external oracle selects one of these two inputs and sends it to M. The oracle decides before the beginning of the interaction whether it sends the first or the second input to M. Importantly, this decision is not known either to Adv or M. The goal of the adversary is to determine which decision was made by the oracle, while the goal of the mechanism is to return the computed output, e.g., a histogram, such that Adv does not find out which decision was made by the oracle.

6: In the case when neighboring streams belong to $\mathbb{Z}$, $\|x^t_{(L)} - x^t_{(R)}\|_\infty \leq 1$

Game 1: Privacy game $\Pi_{M,Adv}$ for the adaptive continual release model.

Input: Stream length $T \in \mathbb{N}$, side $\in \{L, R\}$ (not known to Adv and M)

for $t \in [T]$ **do**
  Adv outputs type$^t \in \{\texttt{challenge}, \texttt{regular}\}$, where `challenge` is only chosen for exactly one value of t
  if type^t = `regular` **then**
    Adv outputs $x^t \in \mathcal{U}$ which is sent to M
  if type^t = `challenge` **then**
    Adv outputs $(x^t_{(L)}, x^t_{(R)}) \in \mathcal{U}^2$
    $x^t_{(\text{side})}$ is sent to M
  M outputs a_t

More formally the relationship between Adv and M is modeled as a game between adversary Adv and mechanism M, given in Game 1.

Definition 7.3.1 (Differential privacy in the adaptive continual release model [128]) *Given a mechanism M the* view *of the adversary Adv in game $\Pi_{M,Adv}$ (Game 1) consists of Adv's internal randomness, as well as the outputs*

of both Adv and M. Let $V^{(\text{side})}_{M,Adv}$ *denote Adv's view at the end of the game run with input* side $\in \{L, R\}$. *Let* $\mathscr{V}$ *be the set of all possible views. Mechanism M is* (ε, δ)-differentially private in the adaptive continual release model *if, for all adversaries Adv and any* $S \subseteq \mathscr{V}$,

$$\Pr(V^{(L)}_{M,Adv} \in S) \leq e^{\varepsilon} \Pr(V^{(R)}_{M,Adv} \in S) + \delta$$

and

$$\Pr(V^{(R)}_{M,Adv} \in S) \leq e^{\varepsilon} \Pr(V^{(L)}_{M,Adv} \in S) + \delta.$$

We also call such a mechanism adaptively (ε, δ)-differentially private.

[118]: Denisov et al. (2022), "Improved Differential Privacy for SGD via Optimal Private Linear Operators on Adaptive Streams"

Denisov et al. [118] shows that ε-dp under continual release implies ε-dp against an adaptive adversary, which is not true for (ε, δ)-dp in general.

Fact 7.3.1 (Prop 2.1 of [118]) *Every mechanism that is* ε*-differentially private in the continual release model is also* ε*-dp in the adaptive continual release model.*

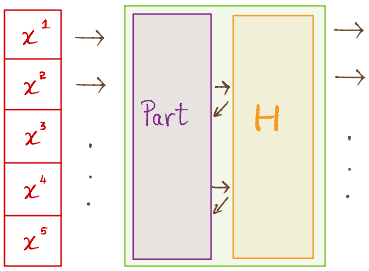
Mechanism for Continual Histogram. A simple mechanism for continual histogram is using d binary counting mechanisms, one per column. With Facts 3.2.5 and 7.3.1, this yields:

Fact 7.3.2 *There is an* ϵ*-dp mechanism for continual histogram in the adaptive model whose* ℓ_∞*-error is bounded by* $O(d \log^2(dT/\beta)\epsilon^{-1})$ *with probability* $1 - \beta$.

This gives a blackbox reduction from ϵ-dp continual histogram to ϵ-dp continual counting. We show in Section 7.7 that any continual histogram mechanism constructed this way must have an error of $\Omega(d \log T)$:

Lemma 7.3.1 *Let A be any* (ϵ/d)*-dp continual counting mechanism. Then the histogram mechanism H, defined by running A independently for each coordinate, must have an error of* $\Omega(d \log(T)\epsilon^{-1})$ *at some time step* $t \leq T$ *with constant probability.*

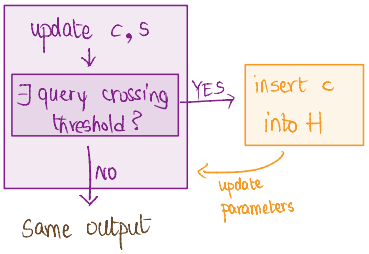
7.4. Histogram Queries Parameterized in Maximum Query Output



Mechanism 7.1 is designed to answer m monotone histogram queries with output-sensitive error on insertion-only streams. It consists of two main parts, a *partitioning* mechanism and a black-box *histogram* mechanism H. The goal of the partitioning mechanism is to sparsify the input stream provided to the histogram mechanism H by partitioning the input stream into *intervals*. It batches consecutive inputs to Mechanism 7.1 together into an interval, and combines these inputs into a single input to H.

In more detail, the algorithm keeps parameters c_i and s_i for each coordinate, which keep the current estimate of the column sum within the current interval, and the total column sum, respectively. On an input x^t, it updates these values (line 10). It then tests if any of the queries on s crosses a threshold (line 17). If not, it returns the output from the previous round.

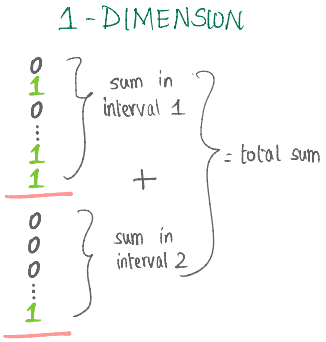
If it does, we insert the c_i values into the blackbox histogram algorithm H (line 18). It then updates the thresholds, the parameters s_i, and the output (lines 25-28). Note that we keep a separate threshold for each query, and they are updated differently depending on whether or not the query answer was close to the threshold in this round (lines 22-23).



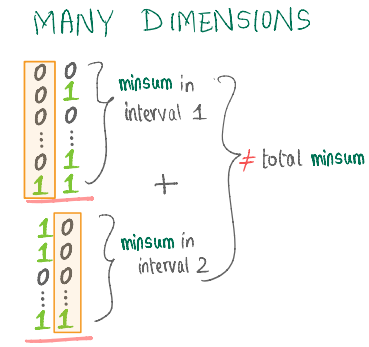
The parameters and thresholds are chosen to minimize the additive error: The longer the intervals, the smaller the error from the histogram mechanism (since it has fewer insertions), and the larger the error *within* an interval (since the same output is used for all time steps within an interval). The parameters of the mechanism (Thresh_k^t, D_j^t, and C_j^t for example) are chosen with the goal of balancing these two kinds of error.

We want to point out two main differences in our approach compared to previous work, which are due to two difficulties: first, we compute non-linear queries on high-dimensional input data, and second, we want to break the $\Omega(d \log T)$ barrier. We explain first the issues that arise and then how we overcome them.

We explain the first difficulty for the query MinSum, but it applies correspondingly to other monotone histogram queries as well. When $d = 1$ (continual counting as in [18]), the partitioning mechanism does not depend on the output of the black-box counting/histogram mechanism. This is because the continual count over a stream is equal to the sum of the continual counts of all intervals. This does not hold for non-linear queries on higher dimensional inputs because the input stream cannot be decomposed into intervals for queries like MinSum, i.e., the MinSum of the entire stream cannot be obtained from knowing just the MinSum value of each interval. Instead, the partitioning algorithm requires an estimate of the current MinSum value at every time step.

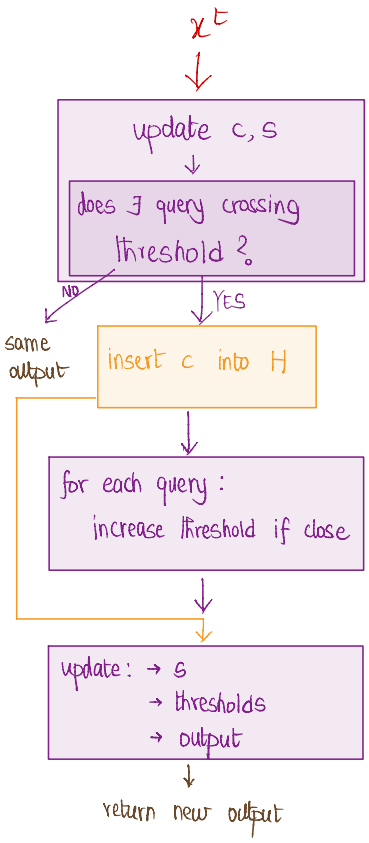


Since we do not want the computation during the current interval to depend on the private data from prior intervals, we reuse the last output of H, as it is a privatized number, in order to keep a running estimate of MinSum. This, however, leads to the following technical challenge: The partitioning mechanism *depends on the outputs of the histogram mechanism of prior intervals*, and the input to the histogram mechanism depends on the output of the partitioning mechanism, and, hence, on the prior output of the histogram mechanism. Furthermore, given two neighboring streams to Mechanism 7.1, the input to the black-box histogram mechanism that is generated by the partitioning mechanism might not necessarily be neighboring streams (consider the case where two wildly different partitions are created on two neighboring streams, leading to inputs to the histogram mechanism that are very far apart). Thus, we cannot use a simple composition theorem to show privacy for the combined mechanism.



To overcome this difficulty, we use a continuous histogram mechanism that is differentially private *even if the inputs are chosen adaptively*. We note that *concurrent* composition theorems as given by, e.g., Haney et al. [133], cannot be used here in a black-box manner, and explain the reasons in more detail in Section 7.8. Adaptive differential privacy of the continuous histogram mechanism allows us to separate the privacy loss incurred by the partitioning mechanism from that of the histogram mechanism.

[133]: Haney et al. (2023), "Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters"



Mechanism 7.1: Mechanism for answering m histogram queries parameterized in the maximum query output.

Input: Stream $x^1, x^2, \ldots \in \{0,1\}^d$, an adaptively ε-differentially private continual histogram mechanism H, failure probability β, additive error bound $\mathrm{err}(t,\beta)$ that holds with probability $\geq 1-\beta$ for the output of H at time step t.

Output: Estimate of $q_k(\mathrm{h}(t))$ for all $k \in [m]$ and all $t \in \mathbb{N}$

`/* Initialization */`

Initialize H

$\beta' \leftarrow 6\beta/\pi^2$, $\beta_t \leftarrow \beta'/t^2$ for any $t \in \mathbb{N}$

$\mathrm{Thresh}_k^1 \leftarrow 3\varepsilon^{-1}(12\ln(2/\beta') + 6\ln(6/\beta') + m\ln(6m/\beta')) + 3\cdot\mathrm{err}(1,\beta'/6)$ for all $k \in [m]$

$c_i \leftarrow 0$ for all $i \in [d]$ ▷ `column sum within interval`

$s_i \leftarrow 0$ for all $i \in [d]$ ▷ `histogram estimate`

$j \leftarrow 1$ ▷ `number of intervals`

$\tau_1 \leftarrow \mathrm{Lap}(6/\varepsilon)$

$\mathrm{out} \leftarrow (q_1(\mathbf{0}), q_2(\mathbf{0}), \ldots, q_m(\mathbf{0}))$ ▷ `current output`

`/* Process the input stream */`

for $t \in \mathbb{N}$ **do**

  $c_i \leftarrow c_i + x_i^t$, $s_i \leftarrow s_i + x_i^t$ for all $i \in [d]$

  `/* Set parameters */`

  $\alpha_\mu^t \leftarrow 12\varepsilon^{-1}\ln(2/\beta_t)$

  $\alpha_\tau^j \leftarrow 6\varepsilon^{-1}\ln(6/\beta_j)$

  $\alpha_\gamma^j \leftarrow 3\varepsilon^{-1}m\ln(6m/\beta_j)$

  $\alpha_H^j \leftarrow \mathrm{err}(j,\beta_j/6)$

  $C_j^t \leftarrow \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$, $D_j^t \leftarrow 3(C_j^t + \alpha_H^j)$

  `/* Test for threshold */`

  $\mu_t \leftarrow \mathrm{Lap}(12/\varepsilon)$

  if $\exists k \in [m] : q_k(s) + \mu_t > \mathrm{Thresh}_k^t + \tau_j$ **then**

    insert $(c_1, \ldots, c_d)$ into H ▷ `Close the current interval`

    reset $c_i \leftarrow 0$ for all $i \in [d]$

    for $k \in [m]$ **do**

      $\gamma_k^j \leftarrow \mathrm{Lap}(3m/\varepsilon)$

      `/* if` $q_k(s)$ `is close to threshold, increase threshold */`

      if $q_k(s) + \gamma_k^j > \mathrm{Thresh}_k^t - C_j^t$ **then**

        $\mathrm{Thresh}_k^t \leftarrow \mathrm{Thresh}_k^t + D_j^t$

    $j \leftarrow j+1$

    `/* update threshold for the new interval */`

    $\mathrm{Thresh}_k^t \leftarrow \mathrm{Thresh}_k^t - D_{j-1}^t + D_j^t \ \forall k \in [m]$

    $\tau_j \leftarrow \mathrm{Lap}(6/\varepsilon)$ ▷ `pick fresh noise`

    $(s_1, \ldots, s_d) \leftarrow \mathrm{output}(H)$

    $\mathrm{out} \leftarrow (q_1(s), \ldots, q_m(s))$

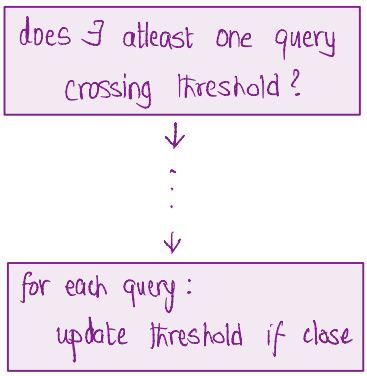
  output out

  $\mathrm{Thresh}_k^{t+1} \leftarrow \mathrm{Thresh}_k^t - D_j^t + D_j^{t+1} \ \forall k \in [m]$

The second difficulty relates to the $\Omega(d \log T)$ barrier. A naïve partitioning technique is to maintain independent thresholds for each query, and to use the sparse vector technique (SVT) separately for each query to check if the query answer is larger than its threshold. If a query answer is larger than the threshold, an interval is closed and the thresholds are increased accordingly. Since this involves interacting with private data at *each* time step for *every* query, this approach incurs the $\Omega(d \log T)$ barrier of all prior work.

To overcome this, we design a partitioning mechanism that works as follows: it checks first if *there exists* a query that crosses a certain predefined threshold

value (line 17). If not, then the mechanism adds the current input to the batch and *does not close* the current interval. The output that was used for the previous time step is reused.



If there exists a query crossing the threshold, then the mechanism *closes* the current interval, sends the batched input to H, and initializes a new interval. At this point, the mechanism has privately determined that *there exists* a query that crosses the threshold. However, this information is not enough to update the thresholds, since we also need to privately determine the *identities* of all the queries that cross the thresholds, which we do next.

Finally, the mechanism checks each query independently and privately if its threshold needs to be updated (line 22). The parameters are chosen such that at least one query has its threshold updated at the end of each interval.

For the utility proof of Dwork et al. [18], it was enough to show that their choice of threshold was larger than the standard deviation of their Laplace random variables. Due to the interplay between several submechanisms, our utility proof requires a more involved analysis.

Theorem 7.4.1 *Let H be any $(\varepsilon/3)$-dp continual histogram mechanism with $\varepsilon > 0$, and let $q_1, \ldots, q_m$ be any m monotone histogram queries. Mechanism 7.1 satisfies ε-dp. If H is the mechanism from Fact 7.3.2, then on input x, Mechanism 7.1 has additive error*

$$O\left(\left(d \log^2(dmq^*/\beta) + m \log(mq^*/\beta) + \log t\right)\varepsilon^{-1}\right)$$

at all $t \in [T]$ simultaneously with probability $1 - \beta$, where $q^* = \max_{k\in[m]} \max_{t\in[T]} q_k(x^1, x^2, \ldots, x^t)$.

Next, we present the privacy and utility proofs of Mechanism 7.1. We use p_j to denote the time step when the threshold is crossed for the j-th time, with $p_0 = 0$ and the final p_j value be set to T. We call the time steps $[p_{j-1}, p_j)$ as the *j-th interval.*

7.4.1. Privacy

Recall that the main technical challenge to prove privacy of Mechanism 7.1 is the following: The partitioning mechanism (which decides when to close an interval) *depends on the outputs of the histogram mechanism for prior intervals* (unlike by Dwork et al. [18], where the partitioning was independent of the output of the counting mechanism), and the input to the histogram mechanism depend on the output of the partitioning mechanism, and, hence, on the prior output of the histogram mechanism. Furthermore, given two neighboring streams, the input to the histogram mechanism might not necessarily be neighboring, since the input to the histogram depends on the partitioning (consider the case where two wildly different partitions are used on two neighboring streams, leading to inputs to the histogram mechanism that are very far apart). Thus, we cannot use a simple composition theorem to show privacy for the combined mechanism. To overcome this difficulty, we use a continuous histogram mechanism that is differentially private *even if the inputs are chosen adaptively.* We then perform a careful privacy analysis to show that the interaction between the adaptively differentially private continuous histogram mechanism and the partitioning mechanism satisfies privacy. The fact that the continuous histogram mechanism is adaptively private allows us to separate the privacy loss incurred by the partitioning mechanism from that of the histogram mechanism in the analysis.

In this proof, we say that two numbers are e^{ε}-close if they satisfy

$$e^{-\varepsilon} \leq \frac{p}{q} \leq e^{\varepsilon}.$$

Lemma 7.4.2 *Let $\varepsilon > 0$. If H is an $(\varepsilon/3)$-differentially private continual histogram mechanism, then Mechanism 7.1 satisfies ε-differential privacy. This holds independent of the initial setting of $(s_1, \ldots, s_d)$, Thresh_k^t, D_j^t, and C_j^ts.*

Proof. Let x and y be two neighboring streams that differ at time t^*. Notice that the outputs of Mechanism 7.1 at any time step are a post-processing of the interval partitioning and the outputs $(s_1, \ldots, s_d)$ of the histogram mechanism H for each interval. Thus, to argue privacy, we consider a mechanism $\mathscr{A}(x)$ which outputs the interval partitions and outputs of H for each interval with input stream x. Let S be any subset of possible outputs of $\mathscr{A}$. We show

$$\Pr[\mathscr{A}(x) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathscr{A}(y) \in S]$$

The arguments also hold when swapping the identities of x and y since they are symmetric, which gives us the privacy guarantee. Thus we focus on proving the inequality above.

This will help us separate the dependence of the histogram and the partitioning.

We first argue that Mechanism 7.1 acts like an adversarial process in the adaptive continual release model towards the histogram mechanism H. From our assumption on H it then follows that the output of H is $\varepsilon/3$-differentially private. We will combine this fact with an analysis of the modified sparse vector technique (which determines when to close an interval) plus the properties of the Laplace mechanism (which determines when a threshold is updated) to argue that the combined mechanism consisting of the partitioning and the histogram mechanism is ε-differentially private.

Recall that an adversary in the adaptive continual release model is given by a privacy game, whose generic form is presented in Game 1. Due to the complicated interaction between the partitioning and H, the specification of such an adversarial process in our setting is given in Game 2.

This coupling lets us spend zero privacy budget for previous intervals.

The basic idea is as follows: Let t^* be the time step at which x and y differ. Conditioned on identical choices for the random variables before time step t^*, we have that all the intervals that the mechanism creates and also the values that the mechanism (in its role as an adversary) gives to the histogram mechanism, are identical for x and y before time step t^*. These are regular time steps in the game. The value for the first interval ending at or after time t^* can differ and constitutes the challenge step. All remaining intervals lead to regular steps in Game 2.

We use only the first stream to partition the stream, but use both streams when giving the input to the histogram algorithm.

Note that the end of the intervals, i.e., the partitioning of the stream, is computed by the adversary. This partitioning is based on the "noisy" histogram (the s_i values), which are computed from the output of H (which can depend on x and y, depending on side) and the values of the input stream x in the current interval - for *either value* of side, since the adversary does not know side. We denote the adversary with input streams x and y by $Adv(x, y)$, and the corresponding game, Game $\Pi_{H,Adv(x,y)}$. $Adv(x, y)$ does not equal $Adv(y, x)$ since the partitioning depends on only the first stream.

The important observation from this game is that there is only one interval, i.e., only one time step for H, where the adversary outputs two values, and in all other time steps it outputs only one value. Also, at the challenge time step

Game 2: Privacy game $\Pi_{H,Adv(x,y)}$ for the adaptive continual release model and m queries for histogram mechanism H

Input: Streams $x = x^1, x^2, \ldots, x^T \in \{0,1\}^d$ and $y = y^1, y^2, \ldots, y^T \in \{0,1\}^d$ such that x and y are neighboring and differ in time t^*, initial values $s_1, \ldots, s_d$, a stream of values $D_1, D_2, \ldots$, a stream of values $C_1, C_2, \ldots$

$p_0 \leftarrow 0, j \leftarrow 1$
$c_i^x \leftarrow 0$ and $c_i^y \leftarrow 0$ for all $i \in [d]$
ChallengeOver $\leftarrow$ False
$\tau \leftarrow \text{Lap}(6/\varepsilon)$
$\widetilde{D}_{(k)} \leftarrow D_1 + \tau$ for all $k \in [m]$
`/* Process the input streams */`
for $t \in [T]$ **do**
  $c_i^x \leftarrow c_i^x + x_i^t$, $s_i \leftarrow s_i + x_i^t$ for all $i \in [d]$
  $c_i^y \leftarrow c_i^y + y_i^t$ for all $i \in [d]$
  $\mu \leftarrow \text{Lap}(12/\varepsilon)$
  if $\exists k \in [m] : q_k(s) + \mu > \widetilde{D}_{(k)}$ **then**
    $p_j \leftarrow t$
    if $p_j \geq t^*$ and *ChallengeOver* = False **then**
      $\text{type}_j \leftarrow$ `challenge`
      output (c^x, c^y)
      ChallengeOver $\leftarrow$ True
    else
      $\text{type}_j \leftarrow$ `regular`
      output c^x
    for $k \in [m]$ **do**
      $\tilde{q}_k(s) \leftarrow q_k(s) + \text{Lap}(3m/\varepsilon)$
      if $\tilde{q}_k(s) > D_{(k)} - C_j$ **then**
        $D_{(k)} \leftarrow D_{(k)} + D_j$
        $j \leftarrow j + 1$
    $\tau \leftarrow \text{Lap}(6/\varepsilon)$
    $D_{(k)} \leftarrow D_{(k)} - D_{j-1} + D_j$ for all $k \in [m]$
    $\widetilde{D}_{(k)} \leftarrow D_{(k)} + \tau$ for all $k \in [m]$
    reset $c_i^x \leftarrow 0$, $c_i^y \leftarrow 0$ for all $i \in [d]$
    receive $(s_1, \ldots, s_d) \leftarrow H$
$p_j \leftarrow T$

where it sends two values c^x and c^y, these values differ by at most 1. Thus the adversarial process that models the interaction between the partitioning mechanism and H fulfills the condition of the adaptive continual release model. As we assume that H is $\varepsilon/3$-differentially private in that model it follows that for all possible neighboring input streams x and y for $\Pi_{H,Adv(x,y)}$ and all possible sides L and R it holds that

$$\Pr(V^{(L)}_{H,Adv(x,y)} \in S) \leq e^{\varepsilon/3} \cdot \Pr(V^{(R)}_{H,Adv(x,y)} \in S)$$

where we use the definition of a view $V^{(L)}_{H,Adv(x,y)}$ and $V^{(R)}_{H,Adv(x,y)}$ from Definition 7.3.1. The same also holds with the positions of x and y switched and for L and R switched. Since the choice of L/R merely decides whether the counts c^x or c^y are sent by the game to H, we abuse notation and specify directly which count is sent to H, as $V^{(x)}_{H,Adv(x,y)}$ or $V^{(y)}_{H,Adv(x,y)}$.

The view of the adversary in Game $\Pi_{H,Adv(x,y)}$ consists of its internal randomness as well as its outputs and the output of H for the whole game, i.e., at the end of the game. The behavior of $Adv(x,y)$ is completely determined by its inputs consisting of x, y, the outputs of H, the thresholds D_j^t and the

values C_j^t, as well as by the functions q_k and the random coin flips. However, for the privacy analysis only the partitioning and the output of H matter since the output of Mechanism 7.1 only depends on those. Thus, we ignore the other values in the view and say that a view V of the adversary $Adv(x, y)$ in Game $\Pi_{H,Adv(x,y)}$ satisfies $V \in S$, if the partitioning and the streams of $(s_1, \ldots, s_d)$ returned from H for all intervals match the output sequences in S. Let C_j^t and D_j^t be as in the mechanism. Assume Game $\Pi_{H,Adv(x,y)}$ is run with those settings of C_j^t and D_j^t. By the definition of $\Pi_{H,Adv(x,y)}$, we have

$$\Pr(\mathscr{A}(x) \in S) = \Pr(V^{(x)}_{H,Adv(x,y)} \in S), \text{ and}$$
$$\Pr(\mathscr{A}(y) \in S) = \Pr(V^{(y)}_{H,Adv(y,x)} \in S).$$

In short, this comes from partitioning and threshold.

If we assume that

$$\Pr(V^{(x)}_{H,Adv(x,y)} \in S) \leq e^{2\varepsilon/3} \cdot \Pr(V^{(x)}_{H,Adv(y,x)} \in S). \tag{7.1}$$

Then privacy follows, since

$$\begin{aligned} \Pr(\mathscr{A}(x) \in S) &= \Pr(V^{(x)}_{H,Adv(x,y)} \in S) \\ &\leq e^{2\varepsilon/3} \cdot \Pr(V^{(x)}_{H,Adv(y,x)} \in S) \\ &\leq e^{\varepsilon} \cdot \Pr(V^{(y)}_{H,Adv(y,x)} \in S) \\ &= e^{\varepsilon} \cdot \Pr(\mathscr{A}(y) \in S). \end{aligned} \tag{7.2}$$

This removes the dependency on the histogram mechanism, and we focus on the partitioning mechanism.

We now prove Inequality 7.1. When we run $Adv(x, y)$ on side x, the interval partitioning is created according to x and the outputs of H. Also for each interval, the input given to H is based on the counts for x, as we consider side x. When we run $Adv(y, x)$ on side x, then the interval partitioning is created according to y and for each interval we give the counts for x as input to H. Thus in both cases the input given to H is based on the counts for x, and hence, to prove Inequality 7.1, it suffices to show that *when running $Adv(x, y)$ on side x and $Adv(y, x)$ on side x, the probabilities of getting a given partition and thresholds are $e^{2\varepsilon/3}$-close.* To simplify notation, we denote running $Adv(x, y)$ on side x as run(x), and $Adv(y, x)$ on side x as run(y).

Denote the interval that t^* belongs to as the j-th interval. Note that the probabilities of computing any fixed sequence of intervals $[p_0, p_1), \ldots, [p_{j-2}, p_{j-1})$ with $p_{j-1} < t^*$ are the same on both run(x) and run(y), since the streams are equal at all time steps before t^*.

The proof of (A) is similar to the privacy of SVT; (B) is by post-processing of the Laplace mechanism; and (C) is by carefully analyzing the sequences of events and their dependencies.

We argue as follows: (A) fixing a particular time $\lambda > p_{j-1}$, the probability of $p_j = \lambda$ is $e^{\varepsilon/3}$-close on run(x) and run(y); and (B) the probabilities of updating the thresholds, i.e., executing line 22 in Game 2 at time p_j for any subset of $[d]$, is $e^{\varepsilon/3}$-close on run(x) and run(y). Then we show that this implies that (C) all the thresholds Thresh_k^t maintained by adversary are the same at the end of the interval.

Before we prove these statements, (A), (B) and (C) together imply that the probabilities that the j-th interval ends at the same time *and* that the thresholds are updated in the same way in all intervals in run (x) and run(y) are $e^{2\varepsilon/3}$-close. This implies that the probabilities $\Pr(V^{(x)}_{H,Adv(x,y)} \in S)$ and $\Pr(V^{(x)}_{H,Adv(y,x)} \in S)$ are $e^{2\varepsilon/3}$-close for any subset S of possible outputs. Thus, Inequality 7.1 and therefore Inequality 7.2 follow, completing the proof.

Statement (A). Fixing a particular time $\lambda > p_{j-1}$, we first show that the probability of interval j ending at λ (i.e., $p_j = \lambda$) is $e^{\varepsilon/3}$-close on run(x) and run(y). Fixing some notation, let $\mu_t \sim \text{Lap}(12/\varepsilon)$ and $\tau_j \sim \text{Lap}(6/\varepsilon)$ be as in the mechanism, let $s^t(x)$ denote the vector of $(s_i)_{i\in[d]}$ at time t for stream x, and f_X denote the density function of the random variable X. For the interval j to close at time λ on run(x), there must exist a $k \in [m]$ with $q_k(s^\lambda(x)) + \mu_\lambda > \text{Thresh}_k^t + \tau_j$ at time λ, and $q_\ell(s^t(x)) + \mu_t \le \text{Thresh}_\ell^t + \tau_j$ for all $p_{j-1} < t < \lambda$ and $\ell \in [m]$.

Note that conditioning on all the random variables being the same on x and y before p_{j-1}, we have that any s_ℓ at time $t \le p_j$ can differ by at most 1 on x and y. Therefore $q_\ell(s^t(x))$ and $q_\ell(s^t(y))$ can also differ by at most 1 by sensitivity of q_ℓ. Therefore, for $p_{j-1} < t < \lambda$, any $\ell \in [m]$ and any fixed value $z \in \mathbb{R}$ that τ_j can take, we have

This is by looking at the cdf of μ_t.

$$\begin{aligned}&\Pr[q_\ell(s^t(x)) + \mu_t \le \text{Thresh}_\ell^t + z]\\ \le &\Pr[q_\ell(s^t(y)) + \mu_t \le \text{Thresh}_\ell^t + z + 1]\end{aligned}$$

Also, for fixed $z \in \mathbb{R}$ (resp. $c \in \mathbb{R}$) that τ_j (resp. μ_λ) can take,

Similarly, by the cdf of μ_λ.

$$\begin{aligned}&\Pr[q_k(s^\lambda(x)) + c > \text{Thresh}_k^t + z]\\ \le &\Pr[q_k(s^\lambda(y)) + c + 2 > \text{Thresh}_k^t + z + 1].\end{aligned}$$

Now, since $\tau_j \sim \text{Lap}(6/\varepsilon)$, we have $f_{\tau_j}(z) \le e^{\varepsilon/6} f_{\tau_j}(z+1)$. Similarly, since $\mu_\lambda \sim \text{Lap}(12/\varepsilon)$, we have $f_{\mu_\lambda}(c) \le e^{2\varepsilon/12} f_{\mu_\lambda}(c+2) = e^{\varepsilon/6} f_{\mu_\lambda}(c)$. Now, integrating over the distributions of τ_j and μ_λ and using these properties gives

$$\Pr[p_j = \lambda \text{ on } x] \le e^{\varepsilon/3} \cdot \Pr[p_j = \lambda \text{ on } y].$$

We conclude that the probability of $p_j = \lambda$ is $e^{\varepsilon/3}$-close on run(x) and run(y).

Statement (B). Conditioned on all previous outputs of H being the same and p_j being equal, we argue that the probabilities of updating any subset of thresholds are close for both runs at time p_j. Since $q_k(s^{p_j}(x))$ and $q_k(s^{p_j}(y))$ can differ by at most 1 for each $k \in [m]$, adding $\gamma_k^j \sim \text{Lap}(3m/\varepsilon)$ to every $q_k(s^{p_j}(y))$ in line 21 ensures that the distributions of $q_k(s^{p_j}(x)) + \gamma_k^j$ and $q_k(s^{p_j}(y)) + \gamma_k^j$ are $e^{\varepsilon/3}$-close for all $k \in [m]$ by composition. Since the condition in line 22 only depends on these variables, this implies that the probabilities of updating the threshold (i.e., executing line 22) on any subset of $[m]$ on run(x) and run(y) are $e^{\varepsilon/3}$-close.

This is by the sensitivity condition as earlier.

Statement (C). *Up to interval $j-1$:* We already argued in (A) that conditioned on all random variables being the same on x and y before interval j, the executions of run(x) and run(y) are identical and, thus, all thresholds are updated in the same way. *Interval j and up:* For any $\ell \ge j$ denote by E_ℓ the event that for run(x) and run(y), all the intervals until interval ℓ end at the same time step, all the thresholds Thresh_k^t for $t \le p_\ell$ are identical, and the random variables used after time p_ℓ take the same values on both runs. We will next argue that conditioned on event E_ℓ, event $E_{\ell+1}$ holds. Note that event E_j holds by (B), and by definition, run(x) and run(y) both use the counts from stream x to compute the input for H. Inductively assume that event E_ℓ holds. Event E_ℓ implies that all intervals $\le \ell$ were closed at the same time on both runs and hence the same counts were given as input to H. Since (a) the streams x and y are identical for all $t > p_\ell$, (b) the thresholds

Need to show that all the thresholds are the same, which we do by inducting on the intervals.

and the outputs of H are identical at the end of interval ℓ, and (c) the random variables used after p_ℓ are identical (which follows from event E_ℓ), we have that the $\ell+1$-st interval ends at the same time on both runs, and that the same thresholds are updated, and by the same amount at time $p_{\ell+1}$. This shows that event $E_{\ell+1}$ holds, as required. □

7.4.2. Accuracy

After processing the input at time step t, let h^t be the actual histogram, s^t be the value of s stored by Mechanism 7.1, and q^{*t} be the maximum query value. Suppose t belongs to interval j, i.e., $t \in [p_{j-1}, p_j)$. Since the mechanism outputs $q_k(s^{p_{j-1}})$ at time t, our goal is to bound the additive error $|q_k(\mathrm{h}^t) - q_k(s^{p_{j-1}})|$ at all times $t \in \mathbb{N}$ and for all queries $k \in [m]$. We do this as follows:

Lemma 7.4.3
1. Use Laplace concentration bounds to bound the maximum value attained by the random variables used by the mechanism.

Lemma 7.4.5
2. Show that if query k crosses the threshold Thresh_k^t, then q_k on the true histogram is not too much smaller than the threshold.

Lemma 7.4.8
3. Show that if query k crosses the threshold Thresh_k^t, then q_k on the true histogram is not too much larger than the threshold.

Lemma 7.4.9
4. Bound the number of intervals produced by the mechanism.

Lemma 7.4.10
5. Use all the above to bound the error of the mechanism.

We define the random variables (RVs) $\mu_t, \tau_j, \gamma_k^j$ as in the mechanism. The variables $\alpha_\mu^t, \alpha_\tau^j, \alpha_\gamma^j$ used in the mechanism are defined such that they bound simultaneously with good probability ($\geq 1-\beta$) the corresponding RVs. In the rest of the section, we condition that the bounds hold on the random variables used in the mechanism.

Lemma 7.4.3 (RV Bounds) *There exists a histogram mechanism H such that the following bounds hold simultaneously with probability $\geq 1-\beta$ for all $t, j \in \mathbb{N}$ and $k \in [m]$*

$$|\mu_t| \leq \alpha_\mu^t, \qquad |\tau_j| \leq \alpha_\tau^j, \qquad |\gamma_k^j| \leq \alpha_\gamma^j,$$
$$\max_{t \in [p_{j-1}, p_j)} \|s^t - h^t\|_\infty \leq \alpha_H^j \quad \forall t \in [p_{j-1}, p_j)$$

where $\alpha_\mu^t = 12\varepsilon^{-1}\ln(2/\beta_t)$, $\alpha_\tau^j = 6\varepsilon^{-1}\ln(6/\beta_j)$, $\alpha_\gamma^j = 3\varepsilon^{-1}m\ln(6m/\beta_j)$,
$\alpha_H^j = O(\varepsilon^{-1}d \cdot (\log(j)\log(d/\beta) + (\log j)^{1.5}\sqrt{\log(d/\beta)}))$

Proof. Using Lemma 3.2.2 gives us the first three bounds below:

1. $\mu_t \sim \mathrm{Lap}(12/\varepsilon)$ satisfies $|\mu_t| < 12\varepsilon^{-1}\ln(2/\beta_t)$ with probability $\geq 1 - \beta_t/2$.
2. $\tau_j \sim \mathrm{Lap}(6/\varepsilon)$ satisfies $|\tau_j| < 6\varepsilon^{-1}\ln(6/\beta_j)$ with probability $\geq 1 - \beta_j/6$.
3. $\gamma_k^j \sim \mathrm{Lap}(3m/\varepsilon)$ satisfies $|\gamma_k^j| \leq 3\varepsilon^{-1}m\ln(6m/\beta_j)$ for all $k \in [m]$ simultaneously with probability $\geq 1 - \beta_j/6$.
4. By assumption, the output of H at time p_{j-1} has additive error at most $\mathrm{err}(j, \beta_j/6)$ with probability at least $1 - \beta_j/6$. In particular, the histogram mechanism from Fact 7.3.2 guarantees $\mathrm{err}(j,\beta) = O(\varepsilon^{-1}d \cdot (\log(j)\log(d/\beta) + (\log j)^{1.5}\sqrt{\log(d/\beta)}))$.

By a union bound, all the four bounds hold at every time step with probability at least $1 - \sum_{t=1}^{\infty}\beta_t/2 - \sum_{j=1}^{\infty}\beta_j/2 = 1-\beta$. □

From the final bound above, we get the following lemma which bounds the error of the query values when computed on the noisy histogram s stored by the mechanism.

Lemma 7.4.4 *Assume Lemma 7.4.3 holds. Let $t \in [T]$ be any time step, and suppose $t \in [p_{j-1}, p_j)$. Then for all $k \in [m]$,*

$$|q_k(s^t) - q_k(h^t)| \leq \alpha_H^j.$$

Proof. This follows, since

$$\begin{aligned} |q_k(s^t) - q_k(h^t)| &\leq \|s^t - h^t\|_\infty && \text{(since } q_k \text{ has sensitivity 1)} \\ &\leq \alpha_H^j && \text{(by Lemma 7.4.3)} \end{aligned}$$

as claimed. □

Since our output at time t is $q_k(s^{p_{j-1}})$, our error is $|q_k(h^t) - q_k(s^{p_{j-1}})|$, which we bound as follows:

$$\begin{aligned} |q_k(h^t) - q_k(s^{p_{j-1}})| &\leq |q_k(h^t) - q_k(h^{p_{j-1}})| + |q_k(h^{p_{j-1}}) - q_k(s^{p_{j-1}})| \\ &\leq |q_k(h^t) - q_k(h^{p_{j-1}})| + \alpha_H^j && \text{(by Lemma 7.4.4)} \\ &\leq q_k(h^t) - q_k(h^{p_{j-1}}) + \alpha_H^j, && \text{(since } q_k \text{ and } h \text{ are monotone and } t \geq p_{j-1}) \end{aligned}$$

our accuracy bound reduces to giving an upper bound on $q_k(h^t)$ and a lower bound on $q_k(h^{p_{j-1}})$.

We say k *crosses the threshold at time t* if line 23 of the mechanism is executed for k at time t. Note that then $t = p_j$ for some j. Our lower bound on $q_k(h^{p_j})$ will be based on the fact that k crosses the threshold at time p_j. At time steps where k did not cross the threshold, our upper bound on $q_k(h^t)$ will follow from a complementary argument to the above lower bound.

For an upper bound on $q_k(h^{p_j})$ at time steps when k crosses the threshold, we first show that k did not cross the threshold at time $p_j - 1$ as follows: Let $p_\ell < p_j$ be the last time step before p_j when k crosses the threshold, and never in between p_ℓ and p_j. Then by definition of the mechanism, $\text{Thresh}_k^{p_j} - \text{Thresh}_k^{p_\ell} = D_j^{p_j}$. We use this to show that q_k must have increased by more than 1 between p_ℓ and p_j. The latter fact implies two things: first, that $j \leq mq^*$; second, that k did not cross the threshold at time $p_j - 1$. The latter can be used to get an upper bound on $q_k(h^{p_j-1})$ and, by the 1-sensitivity of q_k, also on $q_k(h^{p_j})$. For the first interval, there does not exist any such p_ℓ where the threshold was crossed previously. For this, we prove an auxiliary lemma that says that $p_1 > 1$, and hence no threshold was crossed at time $p_1 - 1$, and the rest of the analysis follows.

Combining the two gives an upper bound on $q_k(h^t) - q_k(h^{p_{j-1}})$ of $O(D_j^t + \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j)$, which is the crucial bound needed to upper bound $|q_k(h^t) - q_k(s^{p_{j-1}})|$.

Our first lemma shows that whenever k crosses the threshold, the query value on the true histogram is not too small compared to the threshold.

Lemma 7.4.5 (lower bound) *Assume Lemma 7.4.3 holds. Let $k \in [m]$ and*

suppose k crosses the threshold at time $t = p_j$.

$$q_k(\mathrm{h}^{p_j}) \geq \mathrm{Thresh}_k^{p_j} - \left(\alpha_\mu^{p_j} + \alpha_\tau^j + 2\alpha_\gamma^j + \alpha_H^j\right).$$

Proof. This follows from the sensitivity of q_k and the fact that k crosses the threshold at time p_j.

(by Lemma 7.4.4)	$q_k(\mathrm{h}^{p_j}) \geq q_k(s^{p_j}) - \alpha_H^j$
(by definition of α_γ^j)	$\geq q_k(s^{p_j}) + \gamma_k^j - \alpha_\gamma^j - \alpha_H^j$
(since k crosses the threshold)	$\geq \mathrm{Thresh}_k^{p_j} - C_j^{p_j} - \alpha_\gamma^j - \alpha_H^j$
(by definition of $C_j^{p_j}$)	$\geq \mathrm{Thresh}_k^{p_j} - \alpha_\mu^t - \alpha_\tau^j - 2\alpha_\gamma^j - \alpha_H^j$

as required. □

Using the strategy mentioned above, we then prove that the query value on the true histogram is never too large compared to the threshold. Along the way, we also show that every time k crosses the threshold, the query value on the true histogram must increase.

Lemma 7.4.6 *Assume Lemma 7.4.3 holds. Let $k \in [m]$ and suppose k did not cross the threshold at time t. Then*

$$q_k(h^t) < \mathrm{Thresh}_k^t + \left(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j\right).$$

Proof. Since k did not cross the threshold at time t, either the condition in line 17 was false or the condition in line 22 was false for k at time t. Thus, one of the following holds

if line 17 was false:	$q_k(s^t) < \mathrm{Thresh}_k^t + \alpha_\mu^t + \alpha_\tau^j < \mathrm{Thresh}_k^t + \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$
if line 22 was false:	$q_k(s^t) < \mathrm{Thresh}_k^t - C_j^t + \alpha_\gamma^j < \mathrm{Thresh}_k^t + \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$

Combining this with Lemma 7.4.4 gives the required bound. □

Lemma 7.4.7 *Assume Lemma 7.4.3 holds. No interval is closed on the first time step, i.e., $p_1 > 1$.*

Proof. Note that $C_j^t = \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$. Thus, if the condition in line 17 is true at time p_j, then the condition in line 22 is also true for some k. Said differently, whenever we end a segment, there also exists an k such that k crosses the threshold. Using Lemma 7.4.5 with $t = p_1$ gives us that

$$q_k(\mathrm{h}^{p_1}) \geq \mathrm{Thresh}_k^{p_1} - (\alpha_\mu^{p_1} + \alpha_\tau^1 + 2\alpha_\gamma^1 + \alpha_H^1).$$

Note that since $D_1^{p_1} > \alpha_\mu^{p_1} + \alpha_\tau^1 + 2\alpha_\gamma^1 + \alpha_H^1$, this implies $q_k(\mathrm{h}^{p_1}) > 1$. As q_k increases by at most 1 per time step and $q_k(0, \ldots, 0) = 0$, it follows that $p_1 > 1$. □

Lemma 7.4.8 (upper bound) *Assume Lemma 7.4.3 holds. Let $k \in [m]$ and $t \in \mathbb{N}$.*

$$q_k(h^t) < \mathrm{Thresh}_k^t + \left(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j + 1\right).$$

Further, suppose k crosses the threshold at time $t = p_j$. Then denoting by p_ℓ the last time before p_j that k crossed a threshold, it also holds that $p_j - p_\ell > 1$ and $|q_k(h^{p_j}) - q_k(h^{p_\ell})| > 1$.

Proof. If k did not cross the threshold at time t, then the bound follows from Lemma 7.4.6. Thus assume k crosses the threshold at time $t = p_j$. The first part of the claim follows if we show that k did not cross the threshold at time $p_j - 1$, and $p_j - 1 \geq 1$, since then Lemma 7.4.6 holds at time $p_j - 1$ and q_k has sensitivity one. We show the claim by induction over the number of times k crosses the threshold.

Case 1: p_j is the first time k crosses the threshold. Since p_j is the first time k crosses the threshold, clearly, k did not cross the threshold at time $p_j - 1$. Further, Lemma 7.4.7 gives us that $p_j \geq p_1 > 1$ and therefore $p_j - 1 \geq 1$. Using Lemma 7.4.6 with $t = p_j - 1$, and the fact that q_k has sensitivity one gives the required bound.

Case 2: p_j is not the first time k crosses the threshold. Clearly $p_j - 1 \geq 1$ holds in this case. Then let p_ℓ be the last time at which k crosses the threshold before p_j. By induction, we have for p_ℓ that

$$\begin{aligned} q_k(\mathrm{h}^{p_\ell}) &< \mathrm{Thresh}_k^{p_\ell} + \alpha_\mu^{p_\ell} + \alpha_\tau^{\ell} + \alpha_\gamma^{\ell} + \alpha_H^{\ell} + 1 \\ &\leq \mathrm{Thresh}_k^{p_j} - D_j^{p_j} + \alpha_\mu^{p_j} + \alpha_\tau^{j} + \alpha_\gamma^{j} + \alpha_H^{j} + 1 \end{aligned}$$

Since k crosses the threshold at time p_j, Lemma 7.4.5 with $t = p_j$ gives

$$q_k(\mathrm{h}^{p_j}) \geq \mathrm{Thresh}_k^{p_j} - (\alpha_\mu^{p_j} + \alpha_\tau^{j} + 2\alpha_\gamma^{j} + \alpha_H^{j})$$

Putting both these inequalities together, we get

$$\begin{aligned} |q_k(\mathrm{h}^{p_j}) - q_k(\mathrm{h}^{p_\ell})| &> \left(\mathrm{Thresh}_k^{p_j} - (\alpha_\mu^{p_j} + \alpha_\tau^{j} + 2\alpha_\gamma^{j} + \alpha_H^{j}\right) \\ &\quad - \left(\mathrm{Thresh}_k^{p_j} - D_j^{p_j} + (\alpha_\mu^{p_j} + \alpha_\tau^{j} + \alpha_\gamma^{j} + \alpha_H^{j} + 1)\right) \\ &= D_j^{p_j} - \left(2\alpha_\mu^{p_j} + 2\alpha_\tau^{j} + 3\alpha_\gamma^{j} + 2\alpha_H^{j} + 1\right) > 1, \end{aligned}$$

since $D_j^t \geq 3(C_j^t + \alpha_H^j)$ and $C_j^t = \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$. As q_k has sensitivity one, we have $p_j - p_\ell > 1$, and thus, k did not cross the threshold at time $p_j - 1$. Lemma 7.4.6 with $t = p_j - 1$ and the sensitivity of q_k then gives the required bound. □

We use the second part of the above lemma to bound the number of intervals created by the mechanism, where q^{*t} is the max query output at time t.

Lemma 7.4.9 *Assume Lemma 7.4.3 holds. Mechanism 7.1 creates at most mq^{*t} many segments up to time t.*

Proof. Since $C_j^t = \alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j$, whenever the condition in line 17 is true, then the condition in line 22 is also true for some k, i.e., k crosses the threshold. By Lemma 7.4.8, the query value of q_k on the true histogram grows by at least one every time k crosses the threshold. Since q^{*t} bounds the maximum number of times any query answer can increase before time t, there can be at most mq^{*t} many threshold crossings for all $k \in [m]$ combined, and thus the lemma follows. □

Lemma 7.4.10 *Assume Lemma 7.4.3 holds. Let $t \in \mathbb{N}$ be any time step, and suppose $t \in [p_{j-1}, p_j)$. Then Mechanism 7.1 is $\alpha_j^{(t)}$-accurate at time t, where*

$$\alpha_j^{(t)} = O\left(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j\right)$$

In particular, for all $t \in \mathbb{N}$, Mechanism 7.1 is $\alpha^{(t)}$-accurate, where

$$\alpha^{(t)} = O\left(\alpha_\mu^t + \alpha_\tau^{mq^{*t}} + \alpha_\gamma^{mq^{*t}} + \alpha_H^{mq^{*t}}\right).$$

Proof. Once we prove the first part, the second follows from Lemma 7.4.9. Since $t \in [p_{j-1}, p_j)$, the output of the mechanism at time t is $q_k(s^{p_{j-1}})$. Thus the error at time t is

$$\begin{aligned}
|q_k(h^t) - q_k(s^{p_{j-1}})| &\le |q_k(h^t) - q_k(h^{p_{j-1}})| + |q_k(h^{p_{j-1}}) - q_k(s^{p_{j-1}})| \\
&\le |q_k(h^t) - q_k(h^{p_{j-1}})| + \alpha_H^j && \text{(by Lemma 7.4.4)} \\
&\le q_k(h^t) - q_k(h^{p_{j-1}}) + \alpha_H^j && \text{(since } q_k \text{ monotone and } t \ge p_{j_1}\text{)}
\end{aligned}$$

Our task reduces to giving an upper bound on $q_k(h^t)$, and a lower bound on $q_k(h^{p_{j-1}})$. We have two cases depending on whether k has previously crossed a threshold. Let $t_{\text{first}}(k)$ be the first time in the whole input sequence that k crosses the threshold.

Case 1: $t < t_{\text{first}}(k)$. Since the histogram is empty before the first input arrives, $q_k(h^{p_0}) = 0$. Thus

$$\begin{aligned}
q_k(h^t) - q_k(h^{p_{j-1}}) &\le q_k(h^t) - q_k(h^{p_0}) \\
&< \text{Thresh}_k^t + (\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j + 1) && \text{(by Lemma 7.4.8)} \\
&= D_j^t + (\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j + 1) && \text{(since } \text{Thresh}_k^t = D_j^t\text{)} \\
&= O(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j) && \text{(since } D_j^t = 3(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j)\text{)}
\end{aligned}$$

Case 2: $t \ge t_{\text{first}}(k)$. Let p_ℓ be the largest time step before t when k crosses the threshold. Then

$$\begin{aligned}
q_k(h^t) &\le \text{Thresh}_k^t + (\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j + 1) && \text{(by Lemma 7.4.8)} \\
\text{and} \quad q_k(h^{p_\ell}) &\ge \text{Thresh}_k^{p_\ell} - (\alpha_\mu^{p_\ell} + \alpha_\tau^\ell + 2\alpha_\gamma^\ell + \alpha_H^\ell) && \text{(by Lemma 7.4.5)} \\
&\ge \text{Thresh}_k^t - D_j^t - (\alpha_\mu^t + \alpha_\tau^j + 2\alpha_\gamma^j + \alpha_H^j)
\end{aligned}$$

Putting these together, we get

$$\begin{aligned}
q_k(h^t) - q_k(h^{p_{j-1}}) &\le q_k(h^t) - q_k(h^{p_\ell}) \\
&= O(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j) && \text{(since } D_j^t = 3(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j)\text{)}
\end{aligned}$$

which proves the lemma. □

The accuracy proof for Mechanism 7.1 then follows since we show that Lemma 7.4.3 holds for the corresponding values in the mechanism, and plugging them into the above lemma.

Corollary 7.4.11 *Mechanism 7.1 with a histogram mechanism with error* $\mathrm{err}(t, \beta)$ *has error at most*

$$\alpha^{(t)} = O\left(\frac{1}{\varepsilon}(d\,\mathrm{err}(mq^{*t}, \beta/(mq^{*t})^2) + m\log(mq^{*t}/\beta) + \log t\right)$$

at all time steps t *simultaneously with probability at least* $1-\beta$*. In particular, using the histogram mechanism from Fact 7.3.2 has error at most*

$$\alpha^{(t)} = O\left(\left(d\log^2(dmq^*/\beta) + m\log(mq^*/\beta) + \log t\right)\varepsilon^{-1}\right)$$

at all time steps t *simultaneously with probability at least* $1-\beta$*.*

Proof. Lemma 7.4.3 and Lemma 7.4.10 together give us that Mechanism 7.1 is $\alpha^{(t)}$-accurate at time t, where

$$\alpha_\mu^t = O\left(\varepsilon^{-1}\log(2t/\beta)\right), \qquad \alpha_\tau^j = O\left(\varepsilon^{-1}\log(j/\beta)\right),$$
$$\alpha_\gamma^j = O\left(\varepsilon^{-1}m\log(mj/\beta)\right), \quad \alpha_H^j = O(\mathrm{err}(j, \beta/j^2))$$

Since $\alpha^{(t)} = O\left(\alpha_\mu^t + \alpha_\tau^j + \alpha_\gamma^j + \alpha_H^j\right)$ with $j \le mq^{*t}$, this gives the lemma. For the histogram mechanism from Fact 7.3.2,

$$\begin{aligned}\alpha_H^j &= O\left(\varepsilon^{-1}d\cdot\left(\log(j)\log(dj/\beta) + (\log j)^{1.5}\sqrt{\log(dj/\beta)}\right)\right)\\ &= O\left(\varepsilon^{-1}d\log^2(dj/\beta)\right),\end{aligned}$$

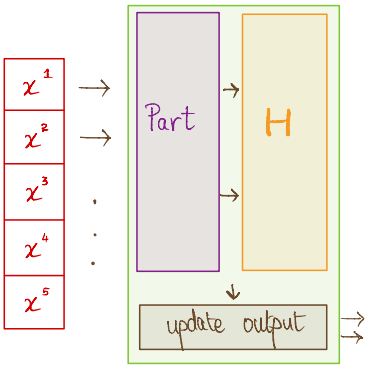
which gives

$$\alpha^{(t)} = O\left(\left(d\log^2(dmq^*/\beta) + m\log(mq^*/\beta) + \log t\right)\varepsilon^{-1}\right)$$

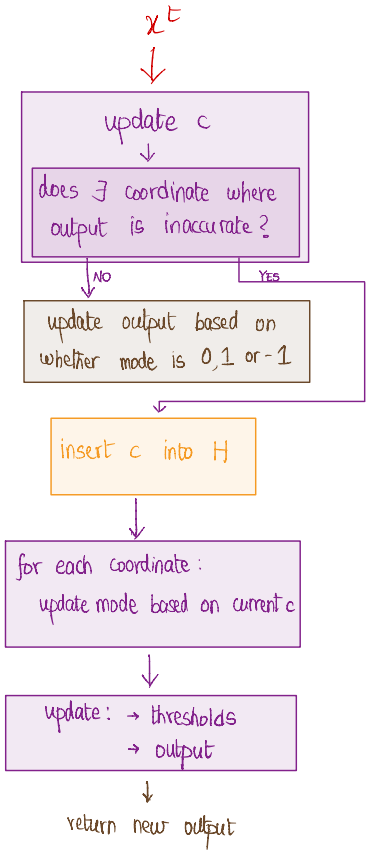
as claimed. □

7.5. Histogram Parameterized in the Number of Fluctuations

Mechanism 7.2 is designed to answer Histogram in the turnstile model with a better error bound when the number of times two consecutive rows differ is small, called the *number of fluctuations*, K. The high-level structure of the mechanism is similar to that of Mechanism 7.1, consisting of a partitioning mechanism that interacts with a black-box histogram mechanism H. The partitioning mechanism batches inputs together into an *interval*, and sends the combined inputs in an interval as a single input to the histogram mechanism, which returns the histogram until that time step.



Earlier, the output of Mechanism 7.1 in an interval was set to the previous output of H, and the output remained unchanged within the interval. In general, when balancing the privacy-accuracy trade-off by limiting the number of time steps for which the private data is accessed, the classic strategy (as by [18]) is to not update the output at all between two updates to the black-box mechanism (here, the mechanism H). We go beyond this paradigm with Mechanism 7.2, by modifying the estimate *even within an interval*. In particular, our histogram estimate is the sum of the last output of H and a function that is linear in the interval length. The function is chosen such that if the input stream remains *stable*, i.e., the value does not change often, we do not need to end the current interval often (here, only $O(K)$ times).



Mechanism 7.2: Mechanism for HISTOGRAM parameterized in the number of fluctuations.

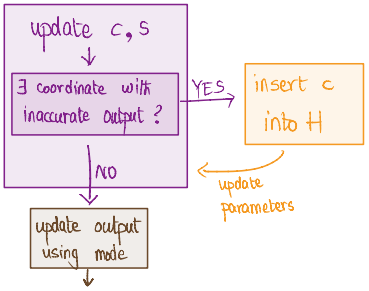
Input: Stream $x^1, x^2, \ldots \in \{-1, 0, 1\}^d$, an $\varepsilon/3$-differentially private continual histogram mechanism H, failure probability β, additive error bound $\mathrm{err}(t, \beta)$ that holds with probability $\geq 1 - \beta$ for the output of H at time step t.

Output: Estimate $\mathrm{h}(t)$ at all $t \in \mathbb{N}$

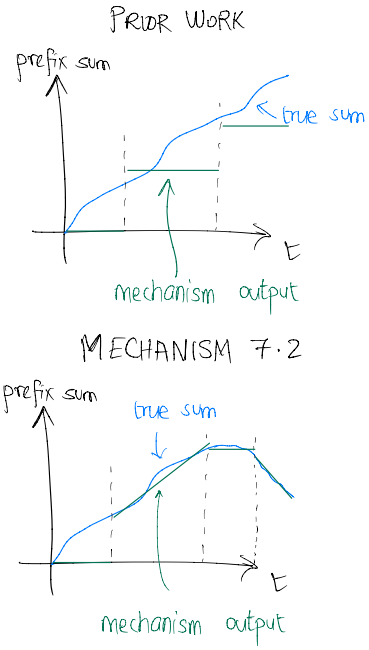
```
/* Initialization of all parameters                                   */
```
Initialize H
$\beta' \leftarrow 6\beta/\pi^2$, $\beta_t \leftarrow \beta'/t^2$ for any $t \in \mathbb{N}$
$j \leftarrow 1$ ▷ number of intervals
$c_i \leftarrow 0$ for all $i \in [d]$ ▷ column sum within interval
$mode_i \leftarrow 0$ for all $i \in [d]$ $t_{\text{diff}} \leftarrow 0$ ▷ length of current interval
$\tau_1 \leftarrow \mathrm{Lap}(9/\varepsilon)$, $\tau_2 \leftarrow \mathrm{Lap}(9/\varepsilon)$
$H_{\text{out}} \leftarrow 0^d$ ▷ initial histogram
```
/* Process the input stream                                           */
```
for $t \in \mathbb{N}$ **do**
  $c_i \leftarrow c_i + x_i^t$ for all $i \in [d]$
  $t_{\text{diff}} \leftarrow t_{\text{diff}} + 1$
  $\alpha_t \leftarrow \frac{27}{\varepsilon}\log(4/\beta_t) + \frac{3d}{\varepsilon}\log(1/\beta_j)$
  $\mathrm{Thresh}_{i,1} \leftarrow mode_i \cdot t_{\text{diff}} - 2\alpha_t$, $i \in [d]$
  $\mathrm{Thresh}_{i,2} \leftarrow mode_i \cdot t_{\text{diff}} + 2\alpha_t$, $i \in [d]$
  $\mu_1^t \leftarrow \mathrm{Lap}(18/\varepsilon)$
  $\mu_2^t \leftarrow \mathrm{Lap}(18/\varepsilon)$
  if $\min_i(c_i - \mathrm{Thresh}_{i,1}) < \tau_1 - \mu_1^t$ **then**
    ```
    /* close the current interval                                     */
    ```
    insert $(c_1, \ldots, c_d)$ into H
    $H_{\text{out}} \leftarrow \mathrm{output}(H)$ ▷ update histogram
    for $i \in [d]$ **do**
      $\lambda_i \leftarrow \mathrm{Lap}(3d/\varepsilon)$
      ```
      /* update modes                                                 */
      ```
      if $c_i + \lambda_i < \mathrm{Thresh}_{i,1} + \alpha_t$ **then**
        $mode_i \leftarrow \max\{mode_i - 1, -1\}$
    reset $c_i \leftarrow 0$ for all $i \in [d]$
    $j \leftarrow j + 1$
    $t_{\text{diff}} \leftarrow 0$; $\tau_1 \leftarrow \mathrm{Lap}(9/\varepsilon)$
  else if $\max_i(c_i - \mathrm{Thresh}_{i,2}) > \tau_2 - \mu_2^t$ **then**
    ```
    /* close the current interval                                     */
    ```
    insert $(c_1, \ldots, c_d)$ into H
    $H_{\text{out}} \leftarrow \mathrm{output}(H)$ ▷ update histogram
    for $i \in [d]$ **do**
      $\lambda_i \leftarrow \mathrm{Lap}(3d/\varepsilon)$
      ```
      /* update modes                                                 */
      ```
      if $c_i + \lambda_i > \mathrm{Thresh}_{i,2} - \alpha_t$ **then**
        $mode_i \leftarrow \min\{mode_i + 1, 1\}$
    reset $c_i \leftarrow 0$ for all $i \in [d]$
    $j \leftarrow j + 1$
    $t_{\text{diff}} \leftarrow 0$; $\tau_2 \leftarrow \mathrm{Lap}(9/\varepsilon)$
  output $H_{\text{out}} + mode \cdot t_{\text{diff}}$

Specifically, the output of Mechanism 7.2 within an interval mimics the behavior of the previous batch of updates. If the histogram was "significantly" increasing (or decreasing) for a coordinate during the previous interval, then the output of Mechanism 7.2 for this coordinate in the current interval adds (or subtracts) an estimate to the last output of H at each time step. This corresponds to guessing the *first-order derivative* of each coordinate of the

histogram within the previous interval and then using this to vary the output of the mechanism for the current interval. When the additive error of the estimate accumulated within an interval crosses a pre-specified threshold for at least one coordinate, the current interval is closed and the guess for the next interval is recomputed based on whether the prior guess overestimated (or underestimated) the true count for the current interval.



The variable c_i tracks the true count for coordinate i within the current interval, and t_{diff} is used to count the number of time steps in the interval. Crucially, we introduce the variables $mode_i$ for $i \in [d]$, which assume values in $\{-1, 0, 1\}$ and are used to guess the slope for each coordinate. Each $mode_i$ is initially 0. Upon an input x^t, we first update c_i and t_{diff} (lines 9-10) and the thresholds (lines 12-13). We use $mode_i \cdot t_{\text{diff}}$ as a guess for the count within the current interval for each coordinate i – that is, if $mode_i = 1$ (or 0 or -1) we guess that coordinate i consists in this interval only of 1's (or 0's or -1's). In each round we check if for some i, the guess is too large or too small compared to c_i (lines 16 and 26). If not, we output the previous histogram output plus $mode_i \cdot t_{\text{diff}}$ for each coordinate i. If for some i, the guess is too large, we insert the c_i values into the blackbox histogram algorithm H and update its output (lines 17-18). We then reduce the modes of all coordinates i where $mode_i \cdot t_{\text{diff}}$ is too large compared to c_i (line 22). If the guess is too small for some i, we perform analogous updates (lines 26-35).



For our parameterized error bound, we analyze it as follows: we subdivide the input stream into maximal contiguous substreams during which the input does not change, called *episodes*, creating at most $K + 1$ episodes in the entire stream. Within each episode, we prove that at most 9 intervals are closed with high probability (whp) as follows: the increase of c_i in one time step, called the *slope of coordinate* i, is constant within an episode and belongs to $\{-1, 0, 1\}$. Thus, within an episode, all d coordinates can be placed into three groups according to their slope. Within each *interval*, the mechanism maintains one of three different modes for each coordinate. We first show that for all coordinates with an identical slope and an identical mode, their modes are updated at the same time step whp. As intervals are only closed when the mode of some coordinate is modified, it suffices to bound the number of time steps within an episode when a mode is updated.

Whenever the mode of a coordinate changes, it holds whp that the absolute difference of its mode and its slope is reduced by 1. Further, the mode will not be updated anymore in this episode when it matches the slope of the coordinate. Thus, if the slope of a group is, say, 1, and there is a subgroup of coordinates of the group with mode -1 at the beginning of the episode, then there will be at most two time steps where the mode of this subgroup changes, namely first to 0 and then to 1. Counting all subgroups and including the potential interval closure when the episode ends results in up to nine intervals closed within an episode whp.

Thus, taking first-order information into account admits improved bounds parameterized in the number of fluctuations while also handling *negative* inputs, and the novel extension of SVT allows the first use of input partitioning techniques in the turnstile model.

Theorem 7.5.1 *Let H be any $(\varepsilon/3)$-dp continual histogram mechanism with $\varepsilon > 0$. Mechanism 7.2 satisfies ε-dp. If H is the mechanism from Fact 7.3.2, then on input x, Mechanism 7.2 has additive error $O((d \log^2(dK/\beta) + \log T)\varepsilon^{-1})$ at all $t \in [T]$ simultaneously with probability $1 - \beta$, where $K = \sum_{t \in [T]} \mathbb{1}(x_t \neq x_{t-1})$.*

Next, we present the privacy and utility proofs of Mechanism 7.2. We use p_j to denote the time step when the threshold is crossed for the j-th time.

7.5.1. Privacy

We prove the following lemma in this section.

Lemma 7.5.2 *Mechanism 7.2 is ε-differentially private.*

To prove Lemma 7.5.2, we note that the outputs of Mechanism 7.2 are a post-processing of three parts:

1. the interval mechanism computing the intervals $p_0, p_1, \dots$,
2. the mode mechanism updating the values of $mode$, and
3. a continual histogram mechanism.

Note that in Mechanism 7.2, the interval and mode mechanisms do *not* depend on the outputs of the continual histogram mechanism, and the histogram mechanism is chosen to be $\varepsilon/3$-differentially private. Thus, it is enough to show that the parts of the mechanism computing $p_0, p_1, \dots$ and the values of $mode$ together are $2\varepsilon/3$-differentially private. Then Mechanism 7.2 is differentially private by the composition theorem (Fact 3.2.5).

Lemma 7.5.3 *The mechanism obtained by running Mechanism 7.2 and outputting only the values of $p_0, p_1, \dots$, and the values of $mode_i$ at every time step, is $2\varepsilon/3$-differentially private.*

We first provide a short proof sketch. The partitioning mechanism consists of instances of AboveThreshold on disjoint parts of the stream, and the updating of modes is a post-processing of a Laplace mechanism. Thus, intuitively, it should be enough to use parallel composition on both the AboveThreshold mechanisms and the Laplace mechanisms, and then use sequential composition on the interval and mode mechanisms above. However, there is a technicality to resolve: the inputs and thresholds to AboveThreshold mechanisms depend on their previous outputs, and the mode updates depend on the output of the AboveThreshold mechanism, which depends on the previous mode update. Thus, we cannot use a regular parallel composition argument and provide the full proof below.

Proof. We focus on proving that the part of Mechanism 7.2 computing the intervals $p_0, p_1, \dots$ and the sequence of $mode$ values is $2\varepsilon/3$-differentially private. Since these do not depend on the outputs of the histogram mechanism, we can then use composition to argue that Mechanism 7.2 is ε-differentially private under continual observation.

As earlier, we are going to couple the random variables until the interval when the streams differ, then induct.

Consider any possible partitioning $P = [p_0, p_1), \dots, [p_{\ell-1}, p_\ell)$ of $[0, T)$ and any stream of $M = M^0, \dots, M^T$, where $M^t = (mode_1^t, \dots, mode_d^t)$ and $mode_i^t$ is the setting of variable $mode_i$ at time t. We show that the probabilities of getting P and M when running Mechanism 7.2 on two neighboring streams are $2\varepsilon/3$-close. For two neighboring streams x and y, let t^* be the time where x and y differ and let $[p_{j-1}, p_j)$ be the interval in P which contains t^*. Note that $\|x^{t^*} - y^{t^*}\|_\infty \leq 1$. Before p_{j-1}, all probabilities are the same; conditioning on the run of x and y to be identical up to time p_{j-1}, we show that the probabilities of closing the next interval of y at p_j and updating the mode of y to $mode_i^{p_j}$ are close to the probabilities of doing so on x. Note that

the original setting of $mode_i$ is always 0 for all i on both x and y. Let $c_i^t(x)$ and $c_i^t(y)$ denote the values of c_i at time t on the run of x and y, respectively.

We compare the probabilities of closing the jth interval at time p_j for x and y, and doing so because the conditions in line 16 resp. 26 were fulfilled. In order to close the j-th interval at time p_j, either the condition in line 16 or 26 has to be fulfilled at time p_j, and neither of them can be fulfilled at any time $t \in (p_{j-1}, p_j)$.

We need to differentiate these two, since the values of $mode$ get updated differently in both cases.

Case A. We analyze the probabilities that at time p_j, the condition in line 16 was fulfilled, and neither conditions in lines 26 and 16 were fulfilled at times $t \in (p_{j-1}, p_j)$. Fix a value z_1 of τ_1 in the interval $(p_{j-1}, p_j]$, z_2 of τ_2 in the interval $(p_{j-1}, p_j]$, and m_1 of $\mu_1^{p_j}$. We have

$$\begin{aligned}
&\text{for all } p_{j-1} < t < p_j\text{:}\\
&\Pr[\min_i c_i^t(x) + \mu_1^t \geq \text{Thresh}_1 + z_1] \leq \Pr[\min c_i^t(y) + \mu_1^t \geq \text{Thresh}_1 + z_1 - 1]\\
&\text{for all } p_{j-1} < t < p_j\text{:}\\
&\Pr[\max_i c_i^t(x) + \mu_2^t \leq \text{Thresh}_2 + z_2] \leq \Pr[\max_i c_i^t(y) + \mu_2^t \leq \text{Thresh}_2 + z_2 + 1]\\
&\text{for } t = p_j\text{:}\\
&\Pr[\min_i c_i^t(x) + m_1 < \text{Thresh}_1 + z_1] \leq \Pr[\min_i c_i^t(y) + m_1 - 2 < \text{Thresh}_1 + z_1 - 1]
\end{aligned}$$

Thus, the same outcome can be achieved by shifting τ_1 and τ_2 by at most 1, and $\mu_1^{p_j}$ by at most 2. By integrating in the same way as in the proof of Lemma 7.4.2, since τ_1 and τ_2 are distributed according to $\text{Lap}(9/\varepsilon)$ and μ_1^t is distributed according to $\text{Lap}(18/\varepsilon)$, the distributions are $\varepsilon/3$-close.

This is similar to the usual analysis of SVT.

Case B. We analyze the probabilities that at time p_j, the condition in line 26 was fulfilled, and neither of the conditions in lines 26 and 16 were fulfilled at times $t \in (p_{j-1}, p_j)$, and the condition in line 16 was not fulfilled at time p_j. As earlier, fix a value z_1 of τ_1 in the interval $(p_{j-1}, p_j]$, z_2 of τ_2 in the interval $(p_{j-1}, p_j]$, and m_2 of $\mu_2^{p_j}$. We have

$$\begin{aligned}
&\text{for all } p_{j-1} < t \leq p_j\text{:}\\
&\Pr[\min_i c_i^t(x) + \mu_1^t \geq \text{Thresh}_1 + z_1] \leq \Pr[\min c_i^t(y) + \mu_1^t \geq \text{Thresh}_1 + z_1 - 1]\\
&\text{for all } p_{j-1} < t < p_j\text{:}\\
&\Pr[\max_i c_i^t(x) + \mu_2^t \leq \text{Thresh}_2 + z_2] \leq \Pr[\max_i c_i^t(y) + \mu_2^t \leq \text{Thresh}_2 + z_2 + 1]\\
&\text{for } t = p_j\text{:}\\
&\Pr[\max_i c_i^t(x) + m_2 > \text{Thresh}_2 + z_2] \leq \Pr[\max_i c_i^t(y) + m_2 + 2 > \text{Thresh}_2 + z_2 + 1].
\end{aligned}$$

Thus, the same outcome can be achieved by shifting τ_1 and τ_2 by at most 1, and $\mu_2^{p_j}$ by at most 2. By integrating in the same way as in the proof of Lemma 7.4.2, since τ_1 and τ_2 are distributed according to $\text{Lap}(9/\varepsilon)$ and μ_2^t is distributed according to $\text{Lap}(18/\varepsilon)$, the distributions are $\varepsilon/3$-close.

Conditioning on ending the j-th interval at p_j and case A (and case B respectively), we need to argue about the updating of the modes (lines 22 and 32, respectively). Since we add $\text{Lap}(3d/\varepsilon)$ to $c_i^{p_j}(x)$ resp. $c_i^{p_j}(y)$ for all i, and $\|c^{p_j}(x) - c^{p_j}(y)\|_1 \leq d$, the probabilities of any output set are $\varepsilon/3$-close by the properties of the Laplace mechanism. By post-processing, the probabilities of updating $mode_i$ to M^{p_j} are $\varepsilon/3$-close on x and y.

Together, the probabilities of getting $[p_0, p_1), \dots, [p_{j-1}, p_j)$ and $M^0, \dots, M^{p_j}$ are $2\varepsilon/3$ close on x and y. Since the rest of the mechanism depends only on p_j, M^{p_j}, and the input streams for times $t > p_j > t^*$, conditioning on p_j and M^{p_j} the probabilities are equal. We get that the probabilities of getting P and M are $2\varepsilon/3$-close on x and y. This shows that the partitioning mechanism together with the mode updates is $2\varepsilon/3$-differentially private. □

7.5.2. Accuracy

Let K_t be the number of times up to time t that two consecutive input data rows differ, even if they differ just in one coordinate, i.e., the number of time such that $x^{t'} \neq x^{t'+1}$ for $t' \leq t$.

Lemma 7.5.4 *With probability at least $1 - 3\beta$, Mechanism 7.2 has error*

$$O\left(\mathrm{err}(9K_t + 9, \beta) + \frac{d}{\varepsilon}\log(9K_t + 9) + \frac{\log(t/\beta)}{\varepsilon}\right)$$

at all time steps t, where $\mathrm{err}(\ell, \beta)$ is the error of the histogram mechanism H that holds with probability at least $1 - \beta$ for all length-ℓ prefixes of the input stream.

Similarly to the accuracy proof in Section 7.4, our proof of Lemma 7.5.4 builds on first bounding the values of all random variables and the error of H such that all bounds hold simultaneously with probability $1 - 3\beta$. We call this event E and condition on it. Formally,

These bounds follow the same lines as in Lemma 7.4.3.

Bound on μ and τ.

With probability $1 - \beta_t/4$, any $Y \sim \mathrm{Lap}(b/\varepsilon)$ satisfies $|Y| \leq b\varepsilon^{-1}\log(4/\beta_t)$. By a union bound, with probability $1 - \beta_t$, we have $|\mu_i^t| \leq 18\varepsilon^{-1}\log(4/\beta_t)$ and $|\tau_i| \leq 9\varepsilon^{-1}\log(4/\beta_t)$ for $i = 1$ and $i = 2$ at any fixed time t. With a union bound over all time steps, and observing that

$$\sum_{t\in[1,T]} \beta_t = \sum_{t\in[1,T]} 6\beta'/(\pi^2 t^2) \leq \beta,$$

it follows that with probability $1 - \beta$ for $i \in \{1, 2\}$ and $t \in [T]$ that $|\mu_i^t| \leq 18\varepsilon^{-1}\log(4/\beta_t)$.

Bound on λ.

With probability $1 - \beta_j$, any $Y \sim \mathrm{Lap}(3d/\varepsilon)$ satisfies $|Y| \leq 3d\varepsilon^{-1}\log(1/\beta_j)$. Thus, by a union bound as above, with probability $1 - \beta$, we have $|\lambda_i| \leq 3d\varepsilon^{-1}\log(1/\beta_j)$ for all values of λ_i.

Bound on H.

By the properties of H, with probability $1 - \beta$, the error of H after j inputs is at most $\mathrm{err}(j, \beta)$ for all j.

Definition 7.5.1 (Event E) *Event E is the event that all three bounds hold.*

The proof now consists of two main lemmata, Lemma 7.5.6 and Lemma 7.5.7. To show them we need:

Claim 7.5.5 *Conditioned on event E the following hold:*

1. *If at some time t the condition in line 16 is true for some $i \in [1, d]$, then the condition in line 21 is true for all $\ell \in [1, d]$ with $c_\ell = c_i$ and $mode_\ell = mode_i$, i.e., $mode_\ell$ is updated for all such ℓ.*
2. *If at some time t the condition in line 26 is true for some $i \in [1, d]$, then the condition in line 31 is true for all $\ell \in [1, d]$ with $c_\ell = c_i$ and $mode_\ell = mode_i$, i.e., $mode_\ell$ is updated for all such ℓ.*

Proof of Claim 7.5.5. If the condition in line 16 or line 26 is true, $t = p_j$ for some j. In the following, we use variable names to denote their value at time t when line 16 is reached. Further, if the condition in line 16 is true, then by the assumed bounds on the random variables in event E, there exists a c_i such that $c_i < \text{Thresh}_{i,1} + \frac{27\log(4/\beta_t)}{\varepsilon}$. Since $|\lambda_i| \leq \frac{3d}{\varepsilon}\log(1/\beta_j)$, we have

$$\begin{aligned} c_i + \lambda_i &\leq c_i + \frac{3d}{\varepsilon}\log(1/\beta_j) \\ &< \text{Thresh}_{i,1} + \frac{27\log(4/\beta_t)}{\varepsilon} + \frac{3d}{\varepsilon}\log(1/\beta_j) && \text{(by event E)} \\ &= \text{Thresh}_{i,1} + \alpha_t. && \text{(by definition of } \alpha_t) \end{aligned}$$

Thus, the condition in line 21 is true for i. Note that this is also case for all $\ell \in [1,d] \setminus i$ with $c_\ell = c_i$ and $mode_\ell = mode_i$, since then $\text{Thresh}_{i,1} = \text{Thresh}_{\ell,1}$. Now, to show that $mode_i$ actually changed, we need to show that $mode_i \neq -1$ at the beginning of the round. Assume $mode_i = -1$ at the beginning of the round. Then

$$c_i + \lambda_i < \text{Thresh}_{i,1} + \alpha_t = -t_{\text{diff}} - \alpha_t,$$

which implies

$$c_i < -t_{\text{diff}} - \alpha_t + \frac{3d}{\varepsilon}\log(1/\beta_j) < -t_{\text{diff}},$$

which is a contradiction since $t_{\text{diff}} \geq c_i \geq -t_{\text{diff}}$ always. By the same argument, $mode_\ell$ is updated for all $\ell \in [1,d]$ with $c_\ell = c_i$ and $mode_\ell = mode_i$. This proves the first part of the claim.,

Similarly, if the condition in line 26 is true, then by the assumed bounds on the random variables, there exists an i such that $c_i > \text{Thresh}_{i,2} - \frac{27\log(4/\beta_t)}{\varepsilon}$. Note that this is also case for all $\ell \in [1,d]$, $j \neq i$ with $c_j = c_i$. Since $|\lambda_i| \leq \frac{3d}{\varepsilon}\log(1/\beta_j)$, we have

$$\begin{aligned} c_i + \lambda_i &\geq c_i - \frac{3d}{\varepsilon}\log(1/\beta_j) \\ &> \text{Thresh}_{i,2} - \frac{27\log(4/\beta_t)}{\varepsilon} - \frac{3d}{\varepsilon}\log(1/\beta_j) && \text{(by event E)} \\ &= \text{Thresh}_{i,2} - \alpha_t. && \text{(by definition of } \alpha_t) \end{aligned}$$

Thus, the condition in line 31 is true for i. Note that this is also case for all $\ell \in [1,d]$, $\ell \neq i$ with $c_\ell = c_i$ and $mode_\ell = mode_i$. Now, to show that $mode_i$ actually changed, we need to show that $mode_i \neq 1$ at the beginning of the round. Suppose $mode_i = 1$. Then

$$c_i + \lambda_i > \text{Thresh}_{i,2} - \alpha_t = t_{\text{diff}} + \alpha_t,$$

implying that

$$c_i > t_{\text{diff}} + \alpha_t - \frac{3d}{\varepsilon}\log(1/\beta_j) > t_{\text{diff}},$$

which is a contradiction since $t_{\text{diff}} \geq c_i \geq -t_{\text{diff}}$ always. By the same argument, $mode_\ell$ is changed, for all $\ell \in [1,d]$ with $c_\ell = c_i$ and $mode_\ell = mode_i$. This proves the second part of the claim. □

The next lemma shows an error bound at time t on the output of Mechanism 7.2 depending on the number of intervals (n_t) produced by the mechanism. This follows from the fact that the histogram receives n_t inputs,

and the fact that we can bound the additional error accumulated within an interval by $O(\varepsilon^{-1} \cdot (d \log(n_t) + \log(t/\beta)))$.

Lemma 7.5.6 *Let n_t be the number of closed intervals at the end of time step t, i.e., if $t = p_j$ for some j, then $n_t = j$, and if $t \in (p_{j-1}, p_j)$, then $n_t = j - 1$. Conditioned on event E, the additive error for all time steps $t' \leq t$ is*

$$O\left(\mathrm{err}(n_t, \beta) + \frac{d}{\varepsilon}\log(n_t) + \frac{\log(t/\beta)}{\varepsilon}\right).$$

Proof. When $t = p_j$ for some $j \leq n_t$, the output is equal to H_{out}, in which case the error is at most $\mathrm{err}(n_t, \beta)$, by the properties of H. Thus we consider the case when $t \in (p_{j-1}, p_j)$, $j \leq n_t + 1$.

Let $t = p_{j-1} + t_{\text{diff}}$ and denote by out^t the output at time t. The error at time step t is given by

$$\max_i \left| \sum_{t'=1}^{t} x_i^{t'} - \text{out}^t \right| \leq \max_i \left(\left| \sum_{t'=1}^{p_{j-1}} x_i^{t'} - \text{out}^{p_{j-1}} \right| + |c_i - mode_i \cdot t_{\text{diff}}| \right),$$

where c_i, $mode_i$ and t_{diff} correspond to the values of those variables in the mechanism at time t. The first term on the right is bounded by $\mathrm{err}(n_t, \beta)$ by the properties of H. For the second term, if either the condition on line 26 or 16 would have been true, then $t = p_j$ for some j, a contradiction. Thus, both conditions were false and the lemma follows, since

$$|c_i - mode_i \cdot t_{\text{diff}}| \leq O(\alpha_t). \qquad \square$$

Lemma 7.5.7 *Conditioned on event E, at any time step t, no more than $\min(t, 9K_t + 9)$ intervals were closed at the end of time step t.*

We partition the stream of input rows into *episodes* such that

1. an episode starts at time $t = 1$ and also at time $t' \in [T]$ if row $x^{t'-1}$ and row $x^{t'}$ differ, and
2. an episode ends at the end of the stream and also at time $t' \in [T]$ if row $x^{t'}$ and row $x^{t'+1}$ differ.

Our proof idea is to show that in no episode more than 9 intervals are closed. As there are at most $K_t + 1$ episodes by the definition of episodes, Lemma 7.5.7 will follow. For episodes in which no interval is closed, nothing has to be shown. Thus, we study in the following episodes in which at least one interval is closed. We first show the following claims, which basically show that if the mechanism receives the same row x^t for a "long enough" time period, then eventually it will set the mode vector equal to x^t.

Let c_i^t (resp. $\text{Thresh}_{i,1}^t$ resp. $\text{Thresh}_{i,2}^t$) denote the value of c_i (resp. $\text{Thresh}_{i,1}$ resp. $\text{Thresh}_{i,2}$) after the initial processing of time step t, i.e., in line 16.

Claim 7.5.8 *Let I be an episode in which at least one interval is closed and let t^* be the first time step at which an interval is closed in I. Conditioned on event E, if at any time step $t > t^*$ where $t \in I$ row x^t equals the vector $mode$ then no mode is updated in time step t.*

Proof. Consider a time step $t > t^*$ in I and let t' with $t^* \leq t' < t$ be the last time before t that a mode was changed. Then c_i is reset to 0 at time t', for all

$i \in [d]$. By definition of t', $mode_i$ did not change since t', and by definition of I, x did not change since t'. Thus,

$$c^t = (t - t') \cdot x^t = t_{\text{diff}} \cdot x^t = t_{\text{diff}} \cdot mode.$$

Thus for each $i \in [d]$,

$$c_i^t - \text{Thresh}_{i,1}^t = 2\alpha_t \quad \text{and} \quad c_i^t - \text{Thresh}_{i,2}^t = -2\alpha_t.$$

It follows that

$$\min_i(c_i^t - \text{Thresh}_{i,1}^t) = 2\alpha_t \quad \text{and} \quad \min_i(c_i^t - \text{Thresh}_{i,2}^t) = -2\alpha_t.$$

But since we condition on E, we have that $|\mu_1^t| \leq \frac{18\log(4/\beta_t)}{\varepsilon}$ and $|\tau_1| \leq \frac{9\log(4/\beta_t)}{\varepsilon}$, and it follows that

$$2\alpha_t = \frac{54}{\varepsilon}\log(4/\beta_t) + \frac{6d}{\varepsilon}\log(1/\beta_j) > \tau_1 - \mu_1^t$$

and that

$$-2\alpha_t = -\frac{54}{\varepsilon}\log(4/\beta_t) - \frac{6d}{\varepsilon}\log(1/\beta_j) < \tau_2 - \mu_2^t.$$

Thus, the conditions on lines 16 and 26 cannot hold and the interval does not close at time t. □

Claim 7.5.9 *Let I be an episode in which at least one interval is closed and let t^* be the first time step at which an interval is closed in I. Conditioned on event E, at any time step $t > t^*$ where $t \in I$ and for any $i \in [d]$ if $mode_i < x_i^t$ at the start of t then $mode_i$ will not decrease in time step t and, symmetrically, if $mode_i > x_i^t$ at the start of t then $mode_i$ will not increase in time step t.*

Proof. Consider a coordinate $i \in [d]$ and a time step $t > t^*$ in I and let $t' \in [t^*, t)$ be the last time before t that a mode was changed. Then c_i is reset to 0 at time t'. By definition of t', $mode_i$ did not change since t', and by definition of I, x_i did not change since t'. Thus,

$$c_i^t = (t - t') \cdot x_i^t = t_{\text{diff}} \cdot x_i^t.$$

In particular, this gives that $x_i^t > mode_i \implies c_i^t \geq t_{\text{diff}} \cdot mode_i$. Thus,

$$\begin{aligned}
c_i^t - \text{Thresh}_{i,1}^t &= c_i^t - t_{\text{diff}} \cdot mode_i + 2\alpha_t && \text{(by definition of Thresh}_{i,1}\text{)} \\
&\geq 2\alpha_t && \text{(since } c_i^t \geq t_{\text{diff}} \cdot mode_i\text{)} \\
&= \frac{27}{\varepsilon}\log(4/\beta_t) + \frac{3d}{\varepsilon}\log(1/\beta_j) + \alpha_t && \text{(by event } E\text{)} \\
&> \alpha_t - \lambda_i
\end{aligned}$$

and thus the condition in line 21 does not hold in time step t and $mode_i$ will not decrease.

We also have that $x_i^t < mode_i \implies c_i^t \leq t_{\text{diff}} \cdot mode_i$. Thus

$$\begin{aligned}
c_i^t - \text{Thresh}_{i,2}^t &= c_i^t - t_{\text{diff}} \cdot mode_i - 2\alpha_t && \text{(by definition of Thresh}_{i,2}\text{)} \\
&\leq -2\alpha_t && \text{(since } c_i^t \leq t_{\text{diff}} \cdot mode_i\text{)} \\
&= -\frac{27}{\varepsilon}\log(4/\beta_t) - \frac{3d}{\varepsilon}\log(1/\beta_j) - \alpha_t && \text{(by event } E\text{)} \\
&< -\alpha_t - \lambda_i
\end{aligned}$$

and, thus, the condition in Line 31 does not hold at time step t and $mode_i$ will not increase. □

The proof of Lemma 7.5.7 now consists of a careful case analysis using Claim 7.5.8 and Claim 7.5.9 to show that within any episode, at most 9 intervals will be closed.

Proof of Lemma 7.5.7. The claim that there are at most t closed intervals follows trivially as at most one interval is closed in a single time step.

We proceed to show that there are at most $9K_t + 9$ closed intervals up to time step t. By Claim 7.5.5 whenever an interval ends, at least one mode has to change. Thus, we will bound the number of time steps that contain a mode update. By the definition of K_t there are exactly $K_t + 1$ many episodes up to time step t. Thus it suffices to show that for any episode I, there are at most 9 time steps in I where a mode is updated.

If no interval is closed in I, i.e., no mode is ever updated, the claim holds trivially for I. Thus in the following assume that there is at least one time step where a mode is updated and let t^* be the first such time step. As a shorthand we use m_i to denote the value of $mode_i$ at the end of time step t^*. Note that $c_i^{t^*} = 0$ and $x_i^t = x_i^{t^*}$ for all $i \in [d]$ and all $t \in I$. Thus for all subsequent time steps $t > t^*$ in I and for all coordinates i, j with $x_i^t = x_j^t$ it holds that $c_i^t = c_j^t$. Thus, by Claim 7.5.5, every time t an interval is closed, there exists an $x^* \in \{-1, 0, 1\}$ and $m^* \in \{-1, 0, 1\}$ such that $mode_i$ is updated for all i with $x_i^t = x^*$ and $m_i = m^*$. As there are only 3 possible values that a variable $x_i^t = x_i^{t^*}$ can assume, it suffices to study for each value $x^* \in \{-1, 0, 1\}$ how often a coordinate i with $x_i^{t^*} = x^*$ updates its mode within I.

Case 1. First we consider all coordinates i with $x_i^{t^*} = -1$ and partition them into 3 subgroups depending on their m_i value. For $m_i = -1$, Claim 7.5.8 shows that there are no time steps with mode updates as the value of the mode and x_i are equal. For $m_i = 0$, Claim 7.5.9 shows that there is at most one time step with mode updates, which decreases the mode to -1. For $m_i = 1$, Claim 7.5.9 shows that there are at most two time steps with mode updates, each decreasing the mode by 1. Thus there are at most 3 time steps with mode updates for all coordinates i with $x_i^{t^*} = -1$.

Case 2. Next consider all coordinates i with $x_i^{t^*} = 0$ and partition them into 3 subgroups depending on m_i. For $m_i = 0$, there are no time steps with mode updates as the value of the mode and x_i are equal. For $m_i = 1$, there is at most one time step with mode updates, which decreases the mode to 0. For $m_i = -1$, there is at most one time steps with mode updates, which increases the mode to 0. Thus there are at most 2 time steps with mode updates for all coordinates i with $x_i^{t^*} = 0$.

Case 3. Finally we consider all coordinates i with $x_i^{t^*} = 1$ and partition them into 3 subgroups depending on m_i. For $m_i = 1$, Claim 7.5.8 shows that there are no time steps with mode updates as the value of the mode and x_i are equal. For $m_i = 0$, there is at most one time step with mode updates, which increases the mode to 1. For $m_i = -1$, there are at most two time steps with mode updates, each increasing the mode by 1. Thus there are at most 3 time steps with mode updates for all coordinates i with $x_i^{t^*} = 1$.

Combined with the update at time step t^*, a mode update happens in at most 9 time steps in episode I. This concludes the proof. □

Lemma 7.5.4 now follows by Lemma 7.5.6 and Lemma 7.5.7.

7.6. Extensions

In the case of Mechanism 7.1, our bounds also extend to the setting when the entries are natural numbers ($\mathcal{U} = \mathbb{N}$). We produce our bounds for (ε, δ)-dp in Section 7.9, where the linear dependencies on d and m are replaced by $\sqrt{d}$ and $\sqrt{m}$ respectively, achieving a similar improvement over prior work as for ε-dp. In the standard definition, neighboring streams x and y may differ in the *entire* input vector at one time step, i.e., there is one time step t^* such that x^{t^*} and y^{t^*} could differ in all d coordinates. If x^{t^*} and y^{t^*} may only differ in up to $b < d$ coordinates, or, more generally, $\|x^{t^*} - y^{t^*}\|_1 \leq b$, then the linear dependency on d is replaced by the same dependency in b.

We summarize Theorem 7.2.1 applied to some widely used query functions.

Corollary 7.6.1 *Let $x = x^1, \ldots, x^T$ be an insertions-only stream. Consider the queries* Histogram, MaxSum, SumSelect, TopK, median, MinSum. *Mechanism 7.1 answers the query at all time steps t, is ε-dp, and has an ℓ_∞-error of $O\left(\left(d \log^2(d\rho/\beta) + \log T\right)\varepsilon^{-1}\right)$ with probability $\geq 1 - \beta$ simultaneously over all time steps, where the parameter ρ is $n_{\min}$ for* MinSum, *n_{median} for* median, *and $n_{\max}$ for the rest, where $n_{\min}$, n_{median}, and $n_{\max}$ are the minimum, median, and maximum column sum.*

As in Mechanism 7.1, the same technique replaces the d term with a $\sqrt{d}$ term for (ε, δ)-dp for Mechanism 7.2 as shown in Section 7.9. Similarly, if two neighboring streams may differ only in up to $b < d$ coordinates at one time step, then the linear dependency in d gets replaced by b.

7.7. An $\Omega(d \cdot \log T)$ Lower Bound for Independently DP Histogram

So far, all upper bounds for Histogram and even for MaxSum which were not polynomial in T relied on running d independent binary counting mechanisms in parallel, and all achieved an error $\Omega(d \log T)$ for ε-differential privacy. In this section we prove that using this strategy one cannot do better. For this, we formally define the following alternative version of differential privacy and neighboring streams of elements from $\{0,1\}^d$:

Definition 7.7.1 (Independent differential privacy) *Let x be a stream from $\mathcal{U} = \{0,1\}^d$. We say x and y are* independently neighboring *if*

$$\forall\, i \in [1,d] \text{ there exists } t_i \in [T] \text{ such that } x_i^t = y_i^t \forall\, t \neq t_i.$$

A mechanism A is independently ε-differentially private *if it is ε-differentially private for independently neighboring x and y.*

This is a superset of the earlier definition of neighboring streams. All x and y which are neighboring are also independently neighboring, but not vice-versa. Independent differential privacy is therefore a stronger property than classic differential privacy.

We show the lower bound using a *packing argument* [134], which relies on the *group privacy* property of differential privacy summarized in Fact 7.7.1.

[134]: Hardt et al. (2010), "On the geometry of differential privacy"

We say x and y are k-neighboring if there exist $x = X_1, X_2, \dots, X_k = y$ such that X_i and X_{i+1} are neighboring for all $1 \le i < k$. In the same way, we say x and y are independently k-neighboring if there exist $x = X_1, X_2, \dots, X_k = y$ such that X_i and X_{i+1} are independently neighboring for all $1 \le i < k$.

Fact 7.7.1 *Let A be an ε-(independently) differentially private mechanism and x and y be (independently) k-neighboring. Then for all $S \in \text{range}(A)$*

$$P(A(x) \in S) \le e^{k\varepsilon} \cdot P(A(y) \in S)$$

Note that computing all d column sums by running independent binary counting mechanisms fulfills independent ε-differential privacy. Next we show that $\Omega(d \log T)$ noise is necessary for computing the noisy column sums in every time step while preserving independent ε-differential privacy.

Theorem 7.7.1 *Assume $d \le \sqrt{T}$. Then there is a $T' = T'(\varepsilon)$ such that any independently ε-differentially private mechanism for computing all d column sums cannot be (α, β)-accurate for constant β and $\alpha \le \frac{d \ln T}{16\varepsilon}$ for streams of length $T \ge T'$.*

Assume without loss of generality in the proof that T is a multiple of b, otherwise we can pad the stream with zero vectors.

Proof. Let $b = \frac{d \ln T}{8\varepsilon}$. We start by dividing $[T]$ into T/b blocks of length b,

$$B_1 = [1, b], \quad B_2 = [b+1, 2b], \quad \dots, \quad B_{T/b} = [T-b+1, T].$$

Now, for any vector $v \in [T/b]^d$ define the following stream $x(v)$:

$$x(v)_i^t = \begin{cases} 1 & \text{if } t \in [(v_i - 1) \cdot b + 1, v_i \cdot b] \\ 0 & \text{otherwise} \end{cases}$$

Denote by s_i^t the estimate for the ith column sum output by the mechanism at time t. $S(v)$ includes all outputs such that $s_i^t < b/2$ for all $t \le (v_i - 1) \cdot b$ and $s_i^t > b/2$ for all $t \ge v_i \cdot b$.

For any mechanism A which is $(b, 1/3)$ accurate, we have

$$P(A(x(v)) \in S(v)) \ge \frac{2}{3}.$$

Further, we have that for any $v, v' \in [T/b]^d$, $x(v)$ and $x(v')$ are independently $2b$-neighboring: for every coordinate i, $x(v')_i$ and $x(v)_i$ differ in at most $2b$ time steps. Therefore, by Fact 7.7.1,

$$P(A(x(v)) \in S(v')) \ge \frac{2}{3} \cdot e^{-2b\varepsilon}.$$

Putting this together, we get a contradiction for large enough T, since

$$\begin{aligned}
1 &\ge \sum_{v \in [T/b]^d} \frac{2}{3} \cdot e^{-2b\varepsilon} && \\
&= \frac{2}{3} \cdot \left(\frac{8T\varepsilon}{d \ln T}\right)^d \cdot e^{-\frac{d \ln T}{4}} && \text{(since } \{S(v)\}_{v \in [T/b]^d} \text{ are disjoint)} \\
&\ge \frac{2}{3} \cdot \left(\frac{8\sqrt{T}\varepsilon}{\ln T}\right)^d \cdot \left(\frac{1}{\sqrt[4]{T}}\right)^d && \text{(since } d \le \sqrt{T}) \\
&= \frac{2}{3} \cdot \left(\frac{8\sqrt[4]{T}\varepsilon}{\ln T}\right)^d > 1 &&
\end{aligned}$$

□

7.8. Usage of Concurrent Composition

Our ε-dp result could alternatively be shown as follows: One could use the result of Qiu and Yi [132] to argue adaptive parallel composition for the partitioning mechanism, then Fact 7.3.1 by Denisov et al. [118] to argue that the ε-dp partitioning mechanism in the continual release model is also ε-dp in the adaptive continual release model, and then use the result of Vadhan and Wang [135] to concurrently compose the adaptive partitioning and adaptive histogram mechanisms. However, this would not reduce the technical complexity of the proof, and also not be self-contained.

[118]: Denisov et al. (2022), "Improved Differential Privacy for SGD via Optimal Private Linear Operators on Adaptive Streams"

[135]: Vadhan et al. (2021), "Concurrent Composition of Differential Privacy"

Moreover, this proof strategy does not work for (ε, δ)-dp at all. Adaptive parallel composition in the (ε, δ)-dp setting is an open problem. Guerra-Balboa et al. [136] give an adaptive parallel composition theorem for (ε, δ)-dp, but their result assumes that the partition of the dataset is given beforehand, while we require that the partition of the dataset is also performed adaptively. Further, it is not enough to show that the partitioning mechanism is differentially private, we would need to show that it is *adaptively* private since a general transformation as in the ε-dp case does not exist here, and is only known for specific mechanisms.

[136]: Guerra-Balboa et al. (2024), "Composition in Differential Privacy for General Granularity Notions"

7.9. Extensions to (ε, δ)-Differential Privacy

We will use noise drawn from the Normal distribution for our mechanism. We use the following continual histogram mechanism H introduced by Fichtenberger et al. [123], which achieves an error of $O(\varepsilon^{-1} \log(1/\delta) \log t \sqrt{d \ln(dt)})$ at time step t. Since their mechanism fulfills the conditions of Theorem 2.1 of Denisov et al. [118], the same privacy guarantees hold for their mechanism in the adaptive continual release model.

[123]: Fichtenberger et al. (2023), "Constant Matters: Fine-grained Error Bound on Differentially Private Continual Observation"

Fact 7.9.1 ((ε, δ)-dp continual histogram against an adaptive adversary) *There is an (ε, δ)-differentially private mechanism in the adaptive continual release model for continual histogram that with probability $\geq 1 - \beta$, has error bounded by $O(\varepsilon^{-1} \log(1/\delta) \log t \sqrt{d \ln(dt/\beta)})$ at time t.*

7.9.1. Histogram Queries

We make the following changes to the mechanism to obtain an (ε, δ)-dp mechanism for histogram queries.

1. Initialize an $(\varepsilon/3, \delta/(2e^{2\varepsilon/3}))$-adaptively dp continual histogram mechanism H.
2. Sample $\gamma_k^j \sim N(0, 18k \ln(4e^{2\varepsilon/3}/\delta)/\varepsilon^2)$.
3. Set α_γ^j to $6\varepsilon^{-1}\sqrt{m \ln(12e^{2\varepsilon/3} m/(\delta\beta_j))}$.

Privacy. We detail the changes to the privacy proof from the ε-dp case. As in the ε-dp case, we need to now show that

$$\Pr[\mathcal{A}(x) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(y) \in S] + \delta$$

Since H is $(\varepsilon/3, \delta/(2e^{2\varepsilon/3}))$-adaptively differentially private, we get that

$$\Pr(V^{(x)}_{H,Adv(x,y)} \in S) \leq e^{\varepsilon/3} \Pr(V^{(y)}_{H,Adv(x,y)} \in S) + \delta/(2e^{2\varepsilon/3})$$

and

$$\Pr(V^{(y)}_{H,Adv(x,y)} \in S) \leq e^{\varepsilon/3} \Pr(V^{(x)}_{H,Adv(x,y)} \in S) + \delta/(2e^{2\varepsilon/3}).$$

Thus all we would need to show would be

$$\Pr(V^{(x)}_{H,Adv(x,y)} \in S) \leq e^{2\varepsilon/3} \Pr(V^{(x)}_{H,Adv(y,x)} \in S) + \delta/2, \tag{7.3}$$

since then

$$\begin{aligned} \Pr(\mathscr{A}(x) \in S) &= \Pr(V^{(x)}_{H,Adv(x,y)} \in S) \\ &\leq e^{2\varepsilon/3} \Pr(V^{(x)}_{H,Adv(y,x)} \in S) + \delta/2 \\ &\leq e^{\varepsilon} \Pr(V^{(y)}_{H,Adv(y,x)} \in S) + \delta \\ &= e^{\varepsilon} \Pr(\mathscr{A}(y) \in S) + \delta \end{aligned} \tag{7.4}$$

The partitioning is still $e^{\varepsilon/3}$-close by the same arguments since we use the same random variables as in the ε-dp case. For the thresholds, note that conditioned on all previous outputs of H and p_j being equal, $q_k(s^{p_j}(x))$ and $q_k(s^{p_j}(y))$ can differ by at most 1 for each $k \in [m]$. Thus the L_2 difference between the two vectors is at most $\sqrt{m}$. By Fact 3.2.3 for the Gaussian mechanism, adding $N(0, 18k \ln(4e^{2\varepsilon/3}/\delta)/\varepsilon^2)$ noise to every $q_k(s^{p_j}(y))$ ensures that the distributions of $q_k(s^{p_j}(x))+\gamma^j_k$ and $q_k(s^{p_j}(y))+\gamma^j_k$ are $(e^{\varepsilon/3}, \delta/2e^{2\varepsilon/3})$-close for all $k \in [m]$. Since the condition in line 22 only depends on those, this implies that the probabilities of executing line 22 on any subset of $[m]$ on run(x) and run(y) are $(e^{\varepsilon/3}, \delta/2e^{\varepsilon/3})$-close, as required.

Accuracy. We have that Lemma 7.4.3 holds with $\alpha^t_\mu, \alpha^j_\tau$ as earlier, $\alpha^j_\gamma = 6\varepsilon^{-1}\sqrt{m \ln(12e^{2\varepsilon/3}m/(\delta\beta_j))}$ and $\alpha^j_H = O(\varepsilon^{-1}\log(1/\delta)\log j\sqrt{d \ln(dj/\beta)})$. Thus, by Lemma 7.4.10, the mechanism has error at most

$$\alpha^{(t)} = O\left(\varepsilon^{-1}\log(1/\delta) \cdot \left(\sqrt{d}\log^{3/2}(dmq^{*t}/\beta) + \sqrt{m}\log(mq^{*t}/\beta) + \log t\right)\right)$$

at all time steps t with probability at least $1-\beta$ as required.

7.9.2. Histogram Parameterized in the Number of Fluctuations

We make the following changes to the mechanism to obtain an (ε, δ)-dp mechanism for Histogram parameterized in the number of fluctuations.

1. Initialize an $(\varepsilon/3, \delta/2)$-adaptively dp continual histogram mechanism H.
2. Sample $\gamma^j_i \sim N(0, 18d \ln(4e^{2\varepsilon/3}/\delta)/\varepsilon^2$.
3. Replace $\frac{3d}{\varepsilon}\log(1/\beta_j)$ with $6\varepsilon^{-1}\sqrt{d \ln(4e^{2\varepsilon/3}/\delta\beta_j)}$.

Privacy. We detail the changes to the privacy proof from the ε-dp case.

We will show that the computation of $p_0, p_1, \ldots$ along with the sequence of $mode$ value updates is $(2\varepsilon/3, \delta/2)$-dp. Then the result follows by basic composition with the histogram mechanism.

The partitioning is still $e^{\varepsilon/3}$-close by the same arguments since we use the same random variables as in the ε-dp case. For the modes, note that conditioning on ending the j-th interval at p_j, $\|c^{p_j}(x) - c^{p_j}(y)\|_2 \leq \sqrt{d}$. Thus, by Fact 3.2.3, adding $N(0, 18d \ln(4e^{2\varepsilon/3}/\delta)/\varepsilon^2$ noise to each $c_i^{p_j}$ ensures that the distributions on x and y are $(e^{\varepsilon/3}, \delta/2e^{\varepsilon/3})$-close. Thus both the partitioning and mode updates together are $(2\varepsilon/3, \delta/2)$-dp as required.

Accuracy. Using the histogram mechanism from Fact 7.9.1, and replacing $\frac{d}{\varepsilon}\log(1/\beta_j)$ in the accuracy proofs with $6\varepsilon^{-1}\sqrt{d \ln(4e^{2\varepsilon/3}/\delta\beta_j)}$, we get that the mechanism has error at most

$$O\left(\varepsilon^{-1}\log(1/\delta) \cdot \left(\sqrt{d}\log^{3/2}(dK/\beta) + \log t\right)\right)$$

at all time steps t with probability at least $1 - \beta$.

Privately Counting Distinct Elements

8.

I put everything into my poetry that I should have put into my life, and now it's too late for me to start all over again.

Vladimir Nabokov, *Mary*

8.1. Introduction

Counting distinct elements in a stream is a fundamental data analysis problem that is widely studied [137–141] with many applications, including network analysis [142] and detection of denial of service attacks [143, 144]. If the data includes sensitive information, the essential challenge is to give accurate answers while providing privacy guarantees to the data owners. In the insertions-only model where elements can only be added, counting distinct elements while preserving privacy is well-studied [145–147].

Recent work by Jain, Kalemaj, Raskhodnikova, Sivakumar, and Smith [19] (which was concurrent with an earlier version of the results presented in this chapter, see [148, Section 5]) initiated the study of this problem in the more general turnstile model. They give an algorithm which is *item-level*, (ε, δ)-differentially private and analyze the additive error parameterized in the *maximum flippancy* of any element, i.e., the number of times that the count of an element changes from 0 to above 0 or vice versa. They also give lower bounds which show that the additive error of the algorithm is nearly tight for item-level differential privacy with respect to their parameterization. There is still a gap for event-level differential privacy posed as an open question. Their algorithm is based on instantiations of the binary tree mechanism.

In this chapter, we show that a simple algorithm based on SVT achieves a tight additive error (up to log factors) for item-level pure and approximate differential privacy, with regards to a different parameterization, namely the *total flippancy*, i.e., the sum of the flippancies of all elements. The additive error depends polynomially on the total flippancy with a smaller exponent than the exponent of the maximum flippancy in the additive error of Jain et al. [19]. Thus, if there are few elements in total, or few elements which change their count from 0 to above 0 or vice versa, then our algorithm achieves a better additive error. Additionally, we give a reduction which shows that for a large class of algorithms, including all existing differentially private algorithms for this problem, the lower bound from item-level differential privacy extends to event-level differential privacy. This is a step towards answering the open question posed by Jain et al. [19].

8.2. Preliminaries

Formally, we maintain a multiset of d different items, and our goal is to continually determine, at each time step, how many of the items are currently present at least once in the multiset.

[137]: Flajolet et al. (1985), "Probabilistic Counting Algorithms for Data Base Applications"

[138]: Flajolet et al. (2007), "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm"

[139]: Kane et al. (2010), "An optimal algorithm for the distinct elements problem"

[140]: Karppa et al. (2022), "HyperLogLogLog: Cardinality Estimation With One Log More"

[141]: Wang et al. (2023), "Better Cardinality Estimators for HyperLogLog, PCSA, and Beyond"

[142]: Weedage et al. (2021), "Locating highly connected clusters in large networks with HyperLogLog counters"

[143]: Akella et al. (2003), "Detecting DDoS attacks on ISP networks"

[144]: Clemens et al. (2023), "DDoS Detection in P4 Using HYPERLOGLOG and COUNTMIN Sketches"

[145]: Bolot et al. (2013), "Private decayed predicate sums on streams"

[146]: Epasto et al. (2023), "Differentially Private Continual Releases of Streaming Frequency Moment Estimations"

[147]: Ghazi et al. (2023), "Private Counting of Distinct and k-Occurring Items in Time Windows"

[19]: Jain et al. (2023), "Counting Distinct Elements in the Turnstile Model with Differential Privacy under Continual Observation"

[148]: Henzinger et al. (2023), "Differentially Private Histogram, Predecessor, and Set Cardinality under Continual Observation"

Definition 8.2.1 (Database models) *Given a turnstile universe $\mathcal{U} = \{-1, 0, 1\}^d$, we consider two database spaces in this chapter. The first space, called the general model, consists of all possible streams of length T,*

$$\mathscr{D}_{\text{General}} = \{x \in \mathcal{U}^T\}$$

and the second space, called the "likes" model[1], is restricted to streams where each element is present at most once at any time step,

$$\mathscr{D}_{\text{Likes}} = \left\{ x \in \mathcal{U}^T \;\middle|\; \sum_{j \le t} x_i^j \in \{0, 1\} \ \forall i \in [d],\ \forall t \in [T] \right\}$$

1: The name was chosen as it models the count of "likes" on a social media website, as motivated by Jain et al.[19].

A 1 at coordinate i denotes inserting element i at that time step, and a -1 corresponds to deleting that item. While our upper bounds allow updating multiple items at each time step, our lower bound also holds in the setting where at most one element may be updated at each time step, i.e., for the restricted universe $\{x \in \mathcal{U} \mid \|x\|_0 \le 1\}$, which we call the *singleton universe*.

We also consider two different neighboring definitions, one of which provides stronger different privacy guarantees than the other.

Definition 8.2.2 (Neighboring models) *For a database space $\mathscr{D}$, two streams x and y are said to be event-level neighboring if*

$$\text{there exists } t \in [T], i \in [d] \text{ such that } x_k^j = y_k^j \text{ for all } (j,k) \neq (t,i).$$

and are said to be item-level neighboring if

$$\text{there exists } i \in [d] \text{ such that } x_k^j = y_k^j \text{ for all } k \neq i.$$

Item-level neighboring streams provide a stronger privacy guarantee compared to event-level neighboring streams since the former allows changing all time steps corresponding to a single item.

Finally, we consider the CountDistinct problem, which asks to count the number of items with positive count at each time step.

Problem 4 (CountDistinct) *For a database space $\mathscr{D}$ and an input stream $x \in \mathscr{D}$, the CountDistinct problem is to release*

$$\textsc{CountDistinct}(x)^t = \sum_{i=1}^{d} \mathbb{1}\left(\sum_{j \le t} x_i^j > 0 \right)$$

at each time step $t \in [T]$, where $\mathbb{1}$ is the indicator function.

The accuracy of a mechanism for this problem is measured using the ℓ_∞ norm over all time steps.

We track the existence of an item at a time step for a given input stream using the existence indicator function

$$f^t(x_i) = \mathbb{1}\left(\sum_{t' \le t} x_i^{t'} > 0 \right).$$

This function is 1 if item i belongs to the database at time t, and 0 otherwise.

Our results are parameterized in the *total flippancy* K, which is the total number of times any item switches from a non-zero count to a zero count, or vice versa. Formally,

$$K = \sum_{i=1}^{d} \sum_{t=2}^{T} \mathbb{1}\left(f^t(x_i) \neq f^{t-1}(x_i) \right).$$

8.2.1. Summary of Results

In this chapter, we give new upper and lower bounds for CountDistinct under *item-level* differential privacy parameterized in the total flippancy. Since $\text{CountDistinct}(x)^t = \sum_{i=1}^{d} f^t(x_i)$, it follows that K is an upper bound on the number of changes in $\text{CountDistinct}(x)$ over time. We focus on pure differential privacy for the introduction.

In the "likes"-model, the total flippancy is equal to the total number of updates.

Upper Bounds. As our first main result, we give item-level differentially private algorithms for CountDistinct in the general model (thus also in the "likes"-model). In the following, we state the exact bounds when K given as input to the algorithm. If K is not given as input to the algorithm, the error bounds worsen by at most a $\log^2 K$ factor.

Theorem 8.2.1 *Given $d \in \mathbb{N}$, $\varepsilon > 0$, $\beta \in (0,1)$, and a known upper bound on the total flippancy K, there exists an item-level ε-differentially private mechanism for CountDistinct in the general model with additive error*

$$O\left(\min\left(d, K, \sqrt{\frac{K \ln(T/\beta)}{\varepsilon}}, \frac{T \log(T/\beta)}{\varepsilon}\right)\right)$$

with probability at least $1-\beta$ at all time steps simultaneously.

As our lower bounds (discussed below) show, these bounds are *tight* if K is known and $K \leq T$. If $K > T$, we incur at most an extra $\ln T$ factor, and if K is not known, we incur extra $\ln K$ factors (see Section 8.8).

For approximate dp, the upper bounds are nearly $O(\min\{\sqrt{T}, \sqrt[3]{K}\})$, and are tight up to $\ln T$, $\ln K$ and $\ln(1/\delta)$ factors.

Lower Bounds. We complement our upper bounds by almost tight lower bounds on the additive error which hold for any item-level differentially private algorithm in the "likes"-model. As this is the "more restricted" of the two models, the lower bounds also carry over to the general model. For ε-differential privacy, our lower bound follows from a packing argument.

Theorem 8.2.2 (Simplified version of Theorem 8.6.1) *Given $L \leq T$, there exists an input stream $x \in \mathscr{D}_{\text{Likes}}$ in the "likes"-model with flippancy $K = \Theta(L)$ such that any item-level, ε-differentially private algorithm for CountDistinct must, with constant probability, have additive error*

$$\Omega\left(\min\left(d, K, \sqrt{\frac{K \max(\ln(T/K), 1)}{\varepsilon}}\right)\right).$$

The lower bound above also holds for singleton updates. When multiple updates are allowed, then K could potentially be larger than T. In that case, Theorem 8.6.1 in Section 8.6 shows that for any $T \leq L \leq dT$, there exists a stream with flippancy $K = \Omega(T)$, $K = O(L)$, such that any item-level, ε-differentially private algorithm for CountDistinct must have error

$$\Omega\left(\min\left(d, \frac{T}{\varepsilon}, \sqrt{\frac{L \max(\ln(T/L), 1)}{\varepsilon}}\right)\right)$$

with constant probability. Since $K = O(L)$, this gives a lower bound of

$$\Omega\left(\min\left(d, \frac{T}{\varepsilon}, \sqrt{\frac{K \max(\ln(T/K), 1)}{\varepsilon}}\right)\right).$$

For approximate dp, we can use a similar strategy as by Jain et al. [19] to get nearly $\Omega(\min\{\sqrt{T}, \sqrt[3]{K}\})$ bounds.

Table 8.1.: Comparison of our results (in purple) and the results of Jain et al. [19] for the different models. For simplicity, we consider singleton insertions and omit polynomial factors in $\ln T$, $\ln(1/\delta)$, and ε^{-1}. The bounds marked * hold for *output dependent* algorithms. The last line follow from continual counting of the difference sequence. The superscript min on a multivariate asymptotic notation means that the notation holds for the min of all operands, e.g., $O^{\min}(A, B) = O(\min\{A, B\})$.

	Item ε-dp	**Item (ε, δ)-dp**	**Event ε-dp**	**Event (ε, δ)-dp**
general model		$O^{\min}(\sqrt{w}, T^{1/3})$		$O^{\min}(\sqrt{w}, T^{1/3})$
Jain et al. [19]	$\Omega^{\min}(w, \sqrt{T})$	$\Omega^{\min}(\sqrt{w}, T^{1/3})$		$\Omega^{\min}(\sqrt{w}, T^{1/4})$
"likes"-model		$O^{\min}(\sqrt{w}, T^{1/3})$		$O^{\min}(\sqrt{w}, T^{1/3})$
Jain et al. [19]	$\Omega^{\min}(w, \sqrt{T})$	$\Omega^{\min}(\sqrt{w}, T^{1/3})$		
general model	$O(\sqrt{K})$	$O(K^{1/3})$	$O(\sqrt{K})$	$O(K^{1/3})$
this work	$\Omega(\sqrt{K})$	$\Omega(K^{1/3})$	$\Omega^{\min}(w, \sqrt{K})^*$	$\Omega^{\min}(\sqrt{w}, K^{1/3})^*$
"likes"-model	$O(\sqrt{K})$	$O(K^{1/3})$	$O(\sqrt{K})$	$O(K^{1/3})$
this work	$\Omega(\sqrt{K})$	$\Omega(K^{1/3})$		
"likes"-model			$O(1)$	$O(1)$

Time and Space Complexity. Our main algorithm (which achieves an $O(\sqrt{K \ln T/\varepsilon})$ additive error bound) can be implemented using constant time per update, assuming that drawing from a Laplace distribution takes constant time. Specifically, the total running time is $O(\#\text{updates} + S_K \cdot t_{\text{Lap}})$, where t_{Lap} is the time to draw one Laplace random variable, and

For approximate dp, S_K is chosen to be nearly $O(\sqrt[3]{K})$.

$$S_K = O\left(\sqrt{\frac{K\varepsilon}{\ln T}}\right).$$

The algorithm uses $O(d)$ words of space. The only information it needs to store are the true counts for each item, plus a constant number of words of extra information. This holds even when K is unknown, since we *sequentially* run our known K algorithm with increasing guesses for K.

[19]: Jain et al. (2023), "Counting Distinct Elements in the Turnstile Model with Differential Privacy under Continual Observation"

Comparison to Jain et al. [19]. In recent work, Jain et al. [19] considered the CountDistinct problem with a similar, but different, parameterization. They parameterize the additive error in the *maximum flippancy*,

$$w_x = \max_{i \in [d]} \left(\sum_{t=2}^{T} \mathbb{1}\left(f^t(x_i) \neq f^{t-1}(x_i)\right) \right).$$

The parameters K and w_x are related by $w_x \leq K \leq d \cdot w_x$. They consider only streams with singleton updates and give algorithms for item-level, (ε, δ)-differential privacy in the general model, with an error bound of

This is roughly $\min\left\{\sqrt{w_x}, \sqrt[3]{T}\right\}$.

$$\widetilde{O}\left(\min\left((\sqrt{w_x}\log T + \log^3 T) \cdot \frac{\sqrt{\log(1/\delta)}}{\varepsilon}, \frac{(T\log(1/\delta))^{1/3}}{\varepsilon^{2/3}}, T\right)\right).$$

In comparison, our additive error bounds in this setting are

$$\widetilde{O}\left(\min\left(\frac{(K\ln(1/\delta)\ln^2 T)^{1/3}}{\varepsilon^{2/3}}\right), K\right)$$

Note that $K \leq T$ for singleton updates, and thus, our upper bounds recover their second and third bound up to a $\ln^{2/3} T$ factor. Furthermore, ignoring polynomial factors in $\log T$, $\log(1/\delta)$ and ε^{-1}, their bound is $O(\sqrt{w_x})$ while ours is $O(K^{1/3})$. Thus, if (roughly) $K < w_x^{3/2}$, our algorithm outperforms theirs. Specifically, if $d \leq \sqrt{w_x}$ or if there are only few items with high flippancy, we expect our algorithm to do better. In cases where the flippancy is well-distributed, i.e., many items have a similar flippancy, and $d \geq \sqrt{w_x}$,

we expect their algorithm to perform better.

In terms of space and time complexity, their algorithm, like ours, needs to maintain a count for each element. Thus, the space in terms of words is $\Omega(d)$. On top of that, they run a variant of the binary tree mechanism, which depending on the implementation, uses $\Omega(\log T)$ space[2]. In their final mechanism, they actually run $\log T$ copies of the binary tree mechanism in parallel, bringing their space consumption to $O(d + \log^2 T)$ words. Thus, the space of our algorithm is an additive $\log^2 T$ term better, which can be crucial for large streams. In terms of time complexity, each of the binary tree mechanism needs to draw $\Omega(T \log T)$ independent Laplace noises, thus their time complexity is at least $\Omega(T \cdot \log^2 T \cdot t_{\text{Lap}})$, where t_{Lap} is the time it takes to draw a Laplace noise. Also here, our algorithm is more efficient.

2: One needs to maintain at least one random variable for each level of the binary tree.

In terms of lower bounds, Jain et al. [19] give the following results for streams of maximum flippancy at most w:

For item-level, ε-dp in the "likes"-model, they give a lower bound of

$$\Omega\left(\min\left(\frac{w}{\varepsilon}, \sqrt{\frac{T}{\varepsilon}}, T\right)\right),$$

for item-level (ε, δ)-dp in the "likes"-model, a lower bound of

$$\widetilde{\Omega}\left(\min\left(\frac{\sqrt{w}}{\varepsilon}, \frac{T^{1/3}}{\varepsilon^{2/3}}, T\right)\right),$$

and for event-level (ε, δ)-dp in the general model, a lower bound of

$$\widetilde{\Omega}\left(\min\left(\frac{\sqrt{w}}{\varepsilon}, \frac{T^{1/4}}{\varepsilon^{3/4}}, T\right)\right).$$

Their upper bounds in the item-level setting match their lower bounds up to factors polynomial in $\log T$ and $\log(1/\delta)$. For event-level in the general model, there is a gap for $\sqrt{T} \leq w \leq T^{2/3}$, and closing this gap was posed as an explicit open question by Jain et al. [19].[3] As our second main result, we make a step towards closing this gap, as shown below.

3: Note that this gap only exists if at most one update per time step is allowed - if many (e.g. up to d many) updates are allowed in each time step, then the lower bound proof for event-level privacy in the general model by Jain et al. [19] can be used to show a lower bound of $\Omega(\sqrt{T})$, for constant ε and δ.

Reduction from item-level, "likes"-model to output-dependent event-level, general model. All the upper bounds mentioned so far hold for *item-level* differential privacy. As our upper bounds hold in the general model and our lower bounds hold in the "likes"-model, we can conclude that for item-level privacy, the "likes"-model and the general model are roughly equally hard. Jain et al. [19] arrived at this conclusion as well, albeit with a different parameterization.

However, for *event-level* neighbors, the picture is different: for the "likes"-model, a very simple algorithm gives an error of $O(\log^2(T)/\varepsilon)$ with constant probability. To see this, define the CountDistinct *difference sequence* as

$$\text{diff}^t(x) = \textsc{CountDistinct}(x)^t - \textsc{CountDistinct}(x)^{t-1}$$

for $t > 1$. As can be easily seen, $(\text{diff}^t(x))_{t>1}$ and $(\text{diff}^t(y))_{t>1}$ differ by at most 1 in at most one time step t for any event-level neighboring streams x and y in the "likes"-model. Thus, applying a standard continual counting algorithm gives the claimed error, as shown for "well-behaved" difference sequences in general by Fichtenberger et al. [129].

[129]: Fichtenberger et al. (2021), "Differentially Private Algorithms for Graphs Under Continual Observation"

For event-level differential privacy and the *general model* however, the best known algorithms are the algorithms for item-level differential privacy in this chapter and by Jain et al. [19]. They also present lower bounds for event-level differential privacy in the general model which, however, leave a gap for certain parameter settings. Closing that gap was explicitly posed as an open question by Jain et al. [19].

We make a step towards closing that gap, by noting that all existing differentially private algorithms for the CountDistinct problem in *any* model share the following property: If $\textsc{CountDistinct}(x) = \textsc{CountDistinct}(y)$ for any two input streams x and y, then the output distributions of the algorithms are equal. We call such algorithms *output-determined*. We show that if we only consider output-determined algorithms for CountDistinct, then achieving *event-level differential privacy in the general model is just as hard as item-level differential privacy for the "likes"-model.* Thus our above lower bounds also apply to such algorithms. In particular, this shows that if one were trying to close the gap for event-level differential privacy in the general model, one needs to find an algorithm which does not only depend on the true answers to CountDistinct[4].

In words, any two streams which produce the same true output, will have the same output distributions.

4: To get around this, the algorithm would need to depend on, for example, the history of CountDistinct values or on the values of the items in some manner.

Theorem 8.2.3 (Simplified version of Theorem 8.5.1) *Given $\varepsilon > 0$, $\delta \geq 0$, let $\mathcal{A}_1$ be an event-level, (ε, δ)-differentially private, output-determined algorithm for CountDistinct that works in the general model and has error at most α for streams of length $T+1$ with probability $1-\beta$.*
Then there exists an item-level, $(2\varepsilon, (1+e^{\varepsilon})\delta)$-differentially private algorithm $\mathcal{A}_2$ for CountDistinct that works in the "likes"-model, and has error at most α for streams of length T with probability $1-\beta$.

Generalizations & Applications. While our algorithms are (nearly) tight for the CountDistinct problem, they are not tailored specifically to the problem and work in a more general setting as well. In particular, recall

$$\textsc{CountDistinct}(x)^t = \mathbb{1}\left(\sum_{t' \leq t} x_i^{t'} > 0\right).$$

Now consider any real-valued function Q on input streams $x = x_1, x_2, \ldots$ with $x_i \in \{-1, 0, 1\}$. We use $Q^t(x)$ to denote $Q(x_1, \ldots, x_t)$. Our algorithm works for any such function Q such that the following two conditions are fulfilled:

We give an example of a natural function satisfying these properties below.

- for any x and y which are neighboring, we have for all time steps t,

$$|Q^t(x) - Q^t(y)| \leq 1,$$

- and Q has bounded "flippancy", i.e.,

$$\sum_{t=1}^{T} |Q^t(x) - Q^{t-1}(x)| \leq K.$$

Theorem 8.2.4 *Given $\varepsilon > 0$ and $\delta \in [0, 1)$, let Q be a function satisfying the above properties. Then there exists an item-level ε-differentially private algorithm for computing Q with additive error*

$$O\left(\min\left(K, \sqrt{\frac{K \ln(T/\beta)}{\varepsilon}}, \frac{T \log(T/\beta)}{\varepsilon}\right)\right)$$

at all time steps with probability at least $1-\beta$.

The extension to unknown K also holds, with extra $\ln K$ factors as earlier. Thus, for a continuous function Q which has *maximum* sensitivity 1 *over all time steps*, we get a bound parameterized in the sum of all differences, i.e., the ℓ_1-norm of the difference sequence. While our results hold in the *turnstile model* and the additive error is parameterized by the total flippancy, Fichtenberger et al. [129] gave an ε-differentially private mechanism with additive error $O(\Gamma \log^{3/2} T \log(T/\beta))$ in the *insertions-only* or *deletions-only* setting, where Γ is the *continuous global sensitivity* which is the ℓ_1-norm of the difference sequence of two neighboring inputs.

We can apply our algorithm to the problem considered in Fichtenberger et al. [129] of continuously counting high degree nodes under differential privacy, which counts the number of nodes with degree at least τ, where τ is given and public. For user-level, edge-differential privacy[5], they give a lower bound of $\Omega(n)$. Our algorithm gives new parameterized bounds for this problem: In particular, when choosing

5: here, neighboring streams may differ in all updates of the same edge.

$$Q^t(x) = \frac{\#\text{ of high degree nodes}}{2},$$

Theorem 8.2.4 gives an error bound of roughly $O(\sqrt{K})$, under ε-dp, and roughly $O(K^{1/3})$ under (ε, δ)-dp. Note that K can be as large as T, but for many applications, it could be much smaller: for example, in social networks, it has been shown that the degree distribution follows a power-law distribution, so the set of high-degree nodes only changes infrequently.

8.3. Technical Overview

The main idea of our algorithm is to use SVT on carefully chosen queries and with carefully chosen thresholds. Recall how SVT works: It is given a database x, a sequence of queries, a threshold *Thresh*, and a stopping parameter S. It will process these queries sequentially, and for each of them answer "yes" or "no" depending on whether or not $q(x)$ is approximately (up to an additive error α) above the threshold. It stops after it has answered "yes" S times. For ε-differential privacy, the additive error bound α of SVT is

Recall the pseudocode of SVT, given in Algorithm 3.1.

$$\alpha = O\left(\frac{S \log(q/\beta)}{\varepsilon}\right)$$

with probability $1 - \beta$, and for (ε, δ)-differential privacy it is

$$\alpha = O\left(\frac{\sqrt{S \log(1/\delta)} \log(q/\beta)}{\varepsilon}\right).$$

Our main idea is to note that the total flippancy K can be seen as an upper bound on *the total change in the output*, i.e., the sum of the absolute differences in the output in every time step. Our strategy is as follows: We start by estimating the number of distinct elements at the beginning of the stream. Then, we keep reporting this estimate until a significant change occurs in the true number of distinct elements. We track whether such a change has occurred using the sparse vector technique. Once there has been a significant change, i.e., once the sparse vector technique answers "yes", we update the output. The goal now is to balance the additive error of the sparse vector technique with the error accumulated between updates. The error between updates is roughly *Thresh*; the error of the sparse vector technique is α; and the total change of the output is bounded by K. To balance the two we set

As earlier, we ignore ε^{-1}, $\log(1/\delta)$, $\log q$ and $\log(1/\beta)$ factors in the following discussion.

Thresh $= \Theta(\alpha)$. Furthermore we have to choose S in a way that makes sure that the sparse vector technique does not abort before we have seen the entire stream. We can show that every time our sparse vector technique answers "yes", the change in output has been roughly *Thresh*. Thus it is enough to set $S > K/\mathit{Thresh}$. As mentioned above, for ε-differential privacy α (and, thus, *Thresh*) must depend linearly on S, which implies that S must be chosen to be $\Theta(\sqrt{K})$, giving an additive error of $O(\sqrt{K})$. For (ε, δ)-differential privacy, we have *Thresh*$= \Theta(\alpha) = O(\sqrt{S})$. This implies that $S^{3/2}$ must be $\Theta(K)$, i.e., $S = \Theta(K^{2/3})$. Thus the additive error is $O(K^{1/3})$.

Note that this requires that K is known at the beginning of the algorithm. If K is unknown, we run the above algorithm for exponentially increasing guesses of K ($K = 2, 4, 8$, etc.). In particular, we run the algorithm for a guess of K, and if it terminates preemptively, we double our guess and repeat. Since we do not know beforehand how many instances are needed, in order to make sure the resulting algorithm is still ε-differentially private, we run the j-th instance with privacy parameter $\varepsilon_j = O(\varepsilon/j^2)$, such that $\sum_{j=1}^{\infty} \varepsilon_j \leq \varepsilon$. At the end of the algorithm, $j = \Theta(\ln K)$, therefore we incur an extra $\ln^2 K$ factor in the additive error.

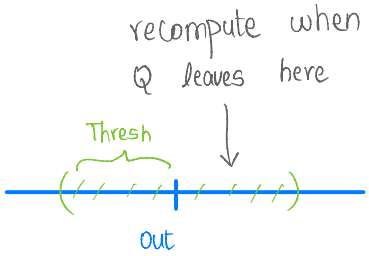
In general, we want to use a sequence $\{a_j\}$ such that $\sum_{j=1}^{\infty}$ is 1, and $1/a_n$ is small asymptotically. We choose $a_j = 1/j^2 \cdot 6/\pi^2$ for this purpose.

8.4. Item-Level Algorithms in General Model

We present our algorithms which work for any input sequence in the general modelin this section. The upper bounds on the additive error for ε-dp match the lower bounds in Section 8.6, except for a $\log(T/\beta)$ factor when $K > T$.

8.4.1. Known Total Flippancy

We prove Theorem 8.4.1 in this section. We give some intuition first on Algorithm 8.1. The algorithm works by iteratively checking if the true number of distinct elements currently present (called Q) is "far" from the current output of our algorithm (called out) using an SVT instantiation. We start the algorithm by estimating out at the beginning of the stream (line 7). Then, we keep outputting out, while we track the difference between out and the true number of distinct elements Q (line 13). Once there has been a significant change, we update the output (line 17).



There are two parameters of interest here. One is the number of times we update the output: we abort after S_K updates happen (line 19). The other is the parameter Thresh, which determines how large the current error needs to be such that we satisfy the condition in line 13. The parameter S_K goes into the error from composition, while the parameter Thresh directly goes into the additive error bound.

The goal is to balance the error accumulated between updates (which is roughly Thresh), and the error from updating out privately (which is roughly S_K for pure differential privacy, and roughly $\sqrt{S_K}$ for approximate differential privacy due to composition). Additionally, we want to make sure our algorithm does not abort before having processed the entire stream. We show that every time SVT returns "yes", the total flippancy in the stream has increased by at least $\Omega(\text{Thresh})$. Since we know the total flippancy is bounded by K, in order to make sure that we do not abort preemptively, we choose S_K such that $S_K \cdot \text{Thresh} \approx K$. Balancing the two error terms yields an additive error of approximately $\sqrt{K}$ for ε-differential privacy, and $K^{1/3}$ for (ε, δ)-differential privacy.

Algorithm 8.1: CountDistinct, known K

Input: Data stream $x = x^1, x^2, \ldots$, initial counts $c_1, \ldots, c_d$ (default 0), parameters ε, δ and β, stream length bound T, stopping parameter $S_K \geq 1$

if $\delta = 0$ **then** $\varepsilon_1 \leftarrow \varepsilon/(2S_K)$
if $\delta > 0$ **then** $\varepsilon_1 \leftarrow \varepsilon/(4\sqrt{2S_K \ln(1/\delta)})$
count $\leftarrow 1$
$\tau_1 \leftarrow \mathrm{Lap}(2/\varepsilon_1)$
$\nu_1 \leftarrow \mathrm{Lap}(1/\varepsilon_1)$
$Q \leftarrow 0$
out $\leftarrow Q + \nu_1$
Thresh $\leftarrow 16\varepsilon_1^{-1} \cdot \ln(2T/\beta)$
for $t = 1, \ldots$ **do**
  $c_i \leftarrow c_i + x_i^t$ for all $i \in [d]$
  $Q \leftarrow |\{i \in [d] \mid c_i > 0\}|$
  $\mu_t \leftarrow \mathrm{Lap}(4/\varepsilon_1)$
  if $|\text{out} - Q| + \mu_t > \text{Thresh} + \tau_{\text{count}}$ **then**
    count $\leftarrow$ count $+ 1$
    $\tau_{\text{count}} \leftarrow \mathrm{Lap}(2/\varepsilon_1)$
    $\nu_{\text{count}} \leftarrow \mathrm{Lap}(1/\varepsilon_1)$
    out $\leftarrow Q + \nu_{\text{count}}$
  output out
  if count $\geq S_K$ **then Abort**

Theorem 8.4.1 *Given $d, T \in \mathbb{N}$, $\varepsilon, \delta, \beta > 0$, and an upper bound on the total flippancy K, there exists*

1. *an item-level ε-differentially private algorithm for the CountDistinct problem in the general model with additive error at most*

$$O\left(\min\left(d, K, \sqrt{\frac{K \ln(T/\beta)}{\varepsilon}}, \frac{T \log(T/\beta)}{\varepsilon}\right)\right)$$

   at all time steps with probability at least $1 - \beta$, and
2. *an item-level (ε, δ)-differentially private algorithm for the CountDistinct problem in the general model with additive error at most*

$$O\left(\min\left(d, K, \left(\frac{K \ln(1/\delta) \ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3}, \frac{\sqrt{T \ln(1/\delta) \log(T/\beta)}}{\varepsilon}\right)\right)$$

   at all time steps with probability at least $1 - \beta$, where $\varepsilon < 1$.

Proof. The $O(\min(d, K))$ bound follows from the fact that the algorithm that outputs 0 at every time step is ε-differentially private and has error at most $\min(d, K)$ for any ε. The third error bounds in the minimum for Theorem 8.4.1 are achieved by Algorithm 8.1, as shown below. Since we assume here all parameters are known, one can compute the minimum of the three bounds and choose the algorithm accordingly. The fourth bound in Theorem 8.4.1 follow by a direct application of the Laplace mechanism Fact 3.2.1 with $\Delta_1 = T$ resp. Gaussian mechanism Fact 3.2.3 with $\Delta_2 = \sqrt{T}$.

The algorithm for our third bound, given in Algorithm 8.1, is based on the sparse vector technique, where S_K is a parameter dependent on K that we choose suitably below. We omit the proof of the following lemma, since it follows from well-known techniques (Sparse Vector Technique, Laplace mechanism and composition theorems).

Lemma 8.4.2 *Given $\varepsilon, \delta \geq 0$, Algorithm 8.1 is ε-differentially private when $\delta = 0$ and Algorithm 8.1 is (ε, δ)-differentially private when $\varepsilon, \delta \in (0, 1)$.*

We show the claimed accuracy bound using the following lemma.

Lemma 8.4.3 *For $\delta = 0$, for any time step t before the algorithm aborts, we have that the maximum error up to time t is at most*

$$O\left(\frac{S_K \ln(T/\beta)}{\varepsilon}\right).$$

Setting $S_K = \sqrt{\frac{K\varepsilon}{18\ln(T/\beta)}} + 1$, with probability at least $1 - \beta$, Algorithm 8.1 does not abort before having seen the entire stream, and has error at most

$$O\left(\sqrt{\frac{K\ln(T/\beta)}{\varepsilon}} + \frac{\ln(T/\beta)}{\varepsilon}\right).$$

For $\delta > 0$, for any time step t before the algorithm aborts, we have that the maximum error up to time t is

$$O\left(\frac{\sqrt{S_K \ln(1/\delta)}\ln(T/\beta)}{\varepsilon}\right).$$

Setting $S_K = \left(\frac{K\varepsilon}{36\sqrt{\ln(1/\delta)\ln(T/\beta)}}\right)^{2/3} + 1$, with probability at least $1 - \beta$, Algorithm 8.1 does not abort before having seen the entire stream, and has error at most

$$O\left(\left(\frac{K\ln(1/\delta)\ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3} + \frac{\sqrt{\ln(1/\delta)}\ln(T/\beta)}{\varepsilon}\right).$$

Proof. Note that at every time step t, we set $Q = \sum_{i=1}^{d} f^t(x_i)$. Let

$$\alpha = \frac{8\ln(2T/\beta)}{\varepsilon_1} = \frac{\text{Thresh}}{2}.$$

We start by bounding all the random variables that arise.

By Laplace tailbounds (Fact 3.2.2), at every time step t:

- $|\tau_\ell| \leq (2/\varepsilon_1)\ln(2T/\beta) = \alpha/4$ with probability at least $1 - \beta/(2T)$, where ℓ is the value of variable count at time step t, and
- $|\mu_t| \leq (4/\varepsilon_1)\ln(2T/\beta) = \alpha/2$ with probability at least $1 - \beta/(2T)$.

Thus, with probability $\geq 1 - \beta$, we have at all time steps t simultaneously:

when the condition in line 13 is true: $$|\text{out} - \sum_{i\in[d]} f^t(x_i)| > \text{Thresh} - \frac{3\alpha}{4} = \frac{5\alpha}{4}$$

when the condition in line 13 is false: $$|\text{out} - \sum_{i\in[d]} f^t(x_i)| \leq \text{Thresh} + \frac{3\alpha}{4} < 3\alpha$$

Further, the random variable ν_ℓ for $\ell \in [S_K]$ is distributed as $\text{Lap}(1/\varepsilon_1)$ and is added to $\sum_{i\in[d]} f^t(x_i)$ at every time step t where out is updated. By the Laplace tail bound (Fact 3.2.2), for all $\ell \in [S_K]$, with probability $1 - \beta$,

$$|\nu_\ell| \leq \frac{\ln(S_K/\beta)}{\varepsilon_1} \leq \frac{\alpha}{8}.$$

Altogether, all of these bounds hold simultaneously with probability at least $1 - 2\beta$. We condition on all these bounds being true.

We then show that the algorithm does not terminate with our choice of S_K.

Assume the algorithm has not terminated yet at time t and let out be the value of variable out at the beginning of time t. Let p_ℓ be the last time step at which the value of out was updated. It holds that $|\text{out} - \sum_{i\in[d]} f_i^{p_\ell}(x)| = |v_\ell| \leq \alpha/8$. If the condition in line 13 is true at time t, then

$$\begin{aligned}
\left|\sum_{i\in[d]} f_i^{p_\ell}(x) - \sum_{i\in[d]} f^t(x_i)\right| &\geq \left|\sum_{i\in[d]} f^t(x_i) - \text{out}\right| - \left|\text{out} - \sum_{i\in[d]} f_i^{p_\ell}(x)\right| \\
&\geq \frac{5\alpha}{4} - \left|\text{out} - \sum_{i\in[d]} f_i^{p_\ell}(x)\right| && \text{(since the condition in line 13 is true)} \\
&\geq \frac{5\alpha}{4} - \frac{\alpha}{8} && \text{(by Laplace tail bounds)} \\
&= \frac{9\alpha}{8}.
\end{aligned}$$

Thus, between two time steps where the value of out is updated, there is a change of at least $9\alpha/8$ in the sum value, i.e., the value of $f^t(x_i)$ has changed at least once for $\geq 9\alpha/8$ different items i. Since the total flippancy is

$$K = \sum_{i=1}^{d}\sum_{t=2}^{T} \mathbb{1}(f^t(x_i) \neq f^{t-1}(x_i)),$$

if we wish to guarantee that the algorithm does not terminate before seeing the entire stream, it suffices to choose $S_K > K/(9\alpha/8)$. For $\delta = 0$, we have

$$\alpha = \frac{8\ln(2T/\beta)}{\varepsilon_1} = \frac{16 S_K \ln(2T/\beta)}{\varepsilon}, \qquad \text{(by choice of } \varepsilon_1\text{)}$$

thus we have to choose S_K to be at least $K\varepsilon/(18 S_K \ln(2T/\beta))$. Choosing

$$S_K = \left\lfloor \sqrt{\frac{K\varepsilon}{18\ln(2T/\beta)}} \right\rfloor + 1$$

fulfills this condition. A similar calculation show that for $\delta > 0$, we should choose

$$S_K = \left(\frac{K\varepsilon}{36\sqrt{\ln(1/\delta)}\ln(T/\beta)}\right)^{2/3} + 1.$$

Now consider any time step t and let out be the output after processing time step t. If the condition in line 13 is false, we showed above that

$$|\text{out} - \sum_{i\in[d]} f^t(x_i)| < 3\alpha.$$

If the condition is true at time t, we have $\text{out} = \sum_{i\in[d]} f^t(x_i) + v_\ell$ for some $\ell \in [S_K]$, and, thus,

$$|\text{out} - \sum_{i\in[d]} f^t(x_i)| \leq \alpha/8 < \alpha.$$

For $\delta = 0$, we have

$$\alpha = \frac{8\ln(2T/\beta)}{\varepsilon_1} = O\left(\sqrt{\frac{K\ln(T/\beta)}{\varepsilon}}\ln(T/\beta) + \frac{\ln(T/\beta)}{\varepsilon}\right).$$

Plugging in the value of S_K yields the final bound for $\delta > 0$. □

To finish the proof of Theorem 8.4.1, note that

$$\frac{\ln(T/\beta)}{\varepsilon} > \sqrt{\frac{K\ln(T/\beta)}{\varepsilon}} \implies \sqrt{\frac{K\ln(T/\beta)}{\varepsilon}} > K.$$

Thus the upper bound $\min(d, K, \sqrt{\varepsilon^{-1}K\ln(T/\beta)})$ holds for $\delta = 0$.

Also, if

$$\frac{\sqrt{\ln(1/\delta)}\ln(T/\beta)}{\varepsilon} > \left(\frac{K\ln(1/\delta)\ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3},$$

then

$$\frac{\sqrt{\ln(1/\delta)}\ln(T/\beta)}{\varepsilon} > K,$$

This can be seen by first cubing the inequality and then dividing by $\varepsilon^{-2}\ln(1/\delta)\ln^2(T/\beta)$.

Thus, for $\delta > 0$, we get an upper bound of

$$\min\left(d, K, \left(\frac{K\ln(1/\delta)\ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3}\right) \qquad \square$$

8.4.2. Generalizations

Let Q be a real-valued function satisfying the properties from earlier[6]. The first bound from Theorem 8.2.4 is achieved by an algorithm that never updates the output, and the third bounds for ε and (ε, δ)-differential privacy are obtained by the Laplace and Gaussian mechanisms, respectively. The second bound for both ε and (ε, δ)-differential privacy is obtained by Algorithm 8.1 by setting $Q = Q^t(x)$ at every time step t. The proofs follow by exchanging $\sum_{i\in[d]} f^t(x_i)$ by $Q^t(x)$ in the proofs of Lemma 8.4.2 and 8.4.3.

6: The properties were:

- for any neighboring x and y, we have for all time steps t that
$$|Q^t(x) - Q^t(y)| \le 1,$$
- and Q has bounded "flippancy", i.e.,
$$\sum_{t=1}^{T} |Q^t(x) - Q^{t-1}(x)| \le K.$$

8.5. A Connection between the General Model under Event-Level Privacy and the "Likes"-Model under Item-Level Privacy

Our bounds as well as the bounds by Jain et al. [19] imply that under *item-level privacy*, the "likes"-model and the general model are roughly equally hard: all upper bounds hold for the general model and all lower bounds hold for the "likes"-model, and the bounds are tight up to a $\log T$ factor. However, under *event-level privacy*, the "likes"-model is significantly easier than the general model: It can be solved via continual counting on the difference sequence of the true output, which gives error polylogarithmic in $\log T$. This is possible because for event-level privacy in the "likes"-model, the difference sequence of the output[7] has ℓ_∞-sensitivity 1 for event-level privacy, but for item-level privacy, the sensitivity can be as large as T.

7: this is the difference between the true output value of the current and the preceding time step.

In the general model, there are no better upper bounds known for event-level differential privacy than for item-level differential privacy, and the upper and lower bounds by Jain et al. [19] for (ε, δ)-differential privacy for the event-level setting in the general model leave a polynomial (in T) gap, in the case where the maximum flippancy $w_x \in (T^{1/2}, T^{2/3})$: In that case, ignoring polynomial factors in ε^{-1}, $\log(1/\delta)$, and $\log T$, the lower bound of Jain et al. [19] is $\Omega(T^{1/4})$, while their algorithm gives an additive error of $O(T^{1/3})$. Specifically, finding the best achievable error for *event-level* privacy in the general model is explicitly posed as an open question by Jain et al. [19].

We resolve this question for a large class of algorithms, called γ-*output-determined* algorithms. All known algorithms for this problem in *any* model are 0-output-determined. Specifically, we show that for γ-output-determined algorithms, our and Jain et al. [19]'s lower bounds for *item-level* privacy in the *"likes"-model* basically carry over to *event-level* privacy in the *general model.* It follows that both algorithms for event-level privacy in the general model are tight up to a factor that is linear in $\log T$ *within the class of output-determined algorithms.* Our reduction works for both pure and approximate dp, and we give the corresponding lower bounds in Theorems 8.6.1 and 8.7.1. In the following, we denote by $\textsc{CountDistinct}(x)$ the stream of true answers to the CountDistinct problem on stream x.

Definition 8.5.1 *Given $\gamma \geq 0$, An algorithm $\mathscr{A}$ for CountDistinct is said to be* γ-output-determined *if for all inputs x and y such that $\textsc{CountDistinct}(x) = \textsc{CountDistinct}(y)$ and any $S \in \text{range}(\mathscr{A})$,*

$$\Pr(\mathscr{A}(x) \in S) \leq \Pr(\mathscr{A}(y) \in S) + \gamma$$

Theorem 8.5.1 *Given $\varepsilon, \delta, \gamma \geq 0$, let $\mathscr{A}_1$ be an event-level, (ε, δ)-differentially private, γ-output-determined algorithm for CountDistinct that works in the general model and has error $\leq \alpha$ for length $T+1$ streams with probability $1-\beta$. Then there exists an item-level, $(2\varepsilon, (1+e^{\varepsilon})\delta + e^{\varepsilon}\gamma)$-differentially private algorithm $\mathscr{A}_2$ for CountDistinct that works in the "likes"-model, and has error $\leq \alpha$ for length T streams with probability $1-\beta$.*

Proof. We describe algorithm $\mathscr{A}_2$, that is item-level $(2\varepsilon, (1+e^{\varepsilon})\delta + e^{\varepsilon}\gamma)$-dp in the "likes"-model, derived from a γ-output-determined algorithm $\mathscr{A}_1$ which is event-level, (ε, δ)-dp in the general model.

Let x be an input stream for CountDistinct in the "likes"-model of length T. Let $x_0 = 0^d \cdot x$, and define for all $t \in [T]$,

We obtain x_0 by attaching a d-dimensional all-zero vector before x.

$$(\mathscr{A}_2(x))^t = (\mathscr{A}_1(x_0))^{t+1}.$$

Note that $\mathscr{A}_1$ can also take inputs from the "likes"-model.

We now show that $\mathscr{A}_2$ is item-level $(2\varepsilon, (1+e^{\varepsilon})\delta + e^{\varepsilon}\gamma)$-differentially private. Let x and y be two item-level neighbouring inputs in the "likes"-model. Thus there exists an item i such that the streams are identical for all items $j \neq i$. Additionally, since we are in the "likes"-model, for any time step t, $\sum_{t' \leq t} x_i^{t'} \in \{0,1\}$ and $\sum_{t' \leq t} y_i^{t'} \in \{0,1\}$.

We will define input streams z and w in the general model such that:

- $\textsc{CountDistinct}(z) = \textsc{CountDistinct}(w)$,
- z is event-level neighbouring to x_0, and
- w is event-level neighbouring to y_0.

This gives us the following: Since $\mathscr{A}_1$ is event-level (ε, δ)-dp and works for the general model, we then have for any $S \in \text{range}(\mathscr{A}_2)$,

$$\begin{aligned}
\Pr[\mathscr{A}_2(x) \in S] &= \Pr[(\mathscr{A}_1(x_0))_{t=2}^{T+1} \in S] \\
&\leq e^{\varepsilon} \Pr[(\mathscr{A}_1(z))_{t=2}^{T+1} \in S] + \delta && \text{(since } x_0 \sim z\text{)} \\
&\leq e^{\varepsilon} \Pr[(\mathscr{A}_1(w))_{t=2}^{T+1} \in S] + \delta + \gamma && \text{(since } \mathscr{A}_1 \text{ is } \gamma\text{-output-determined)} \\
&\leq e^{2\varepsilon} \Pr[(\mathscr{A}_1(y_0))_{t=2}^{T+1} \in S] + (1+e^{\varepsilon})\delta + e^{\varepsilon}\gamma && \text{(since } w \sim y_0\text{)} \\
&= e^{2\varepsilon} \Pr[\mathscr{A}_2(y) \in S] + (1+e^{\varepsilon})\delta + e^{\varepsilon}\gamma,
\end{aligned}$$

as required. To define such z and w, let $-e_i$ be the vector such that $-e_i(j) = 0$ for all $j \neq i$ and $-e_i(i) = -1$. Then define

$$z = -e_i \cdot x \quad \text{and} \quad w = -e_i \cdot y.$$

Note that z and w are valid input streams for the general model, while they are not valid for the "likes"-model[8]. Clearly, z is event-level neighbouring to x_0, and w is event level neighbouring to y. Since $\sum_{t' \leq t} x_i^t \in \{0, 1\}$ for all $t \in [T]$ we have $\sum_{t' \leq t} z_i^t \leq 0$ for all $t \in [T+1]$. By the same argument, we have $\sum_{t' \leq t} w_i^t \leq 0$ for all $t \in [T+1]$. Since z and w only differ in the i-th coordinate, which never contributes to the CountDistinct value as it is never 1, we have $\text{CountDistinct}(z) = \text{CountDistinct}(w)$.

8: They are not valid for the "likes"-model as they begin with item i having count -1.

We are left with analyzing the error of the two algorithms. For this, note that by definition of x_0, we have

$$\text{CountDistinct}(x_0)^{t+1} = \text{CountDistinct}(x)^t.$$

Thus, running $\mathscr{A}_2$ on x gives the same error as running $\mathscr{A}_1$ on x_0. □

In particular, for any output-determined algorithm, Theorem 8.5.1 implies that all lower bounds on the error for the CountDistinct problem under *item-level* differential privacy which hold for the *"likes"-model*[9], carry over to all current algorithms under *event-level* differential privacy in the *general model*. This means that if there is an algorithm achieving a better error than the bounds stated in Theorem 8.7.1 and by Jain et al. [19] for event-level differential privacy in the general model, it cannot be γ-output-determined for $\gamma = O(\delta)$, i.e., it must be such that it does not *only* depend on the number of distinct elements at any given time step.

9: This includes all lower bounds for CountDistinct under item-level differential privacy shown in this chapter in Theorem 8.7.1 and by Jain et al. [19].

8.6. Item-Level Lower Bounds in the "Likes"-Model

We show lower bounds for solving CountDistinct under item-level privacy in the "likes"-model. The lower bounds also apply to the general model.

Theorem 8.6.1 *Given $d, T \in \mathbb{N}$, $\varepsilon > 0$, with $T > 4$,*

1. *given $L \in \mathbb{N}$ with $8 \leq L \leq dT$, there exists an input stream $x \in \mathscr{D}_{\text{likes}}$ with multiple updates per time step, with flippancy K and*

$$\min\left\{\frac{3L}{8}, \frac{T}{4} - 1\right\} \leq K \leq \min\left\{L, \frac{dT}{4}\right\}$$

such that any ε-differentially private algorithm to the CountDistinct problem with item-level privacy with error at most α at all time steps with probability at least $2/3$ must satisfy

$$\alpha = \Omega\left(\min\left(d, L, \frac{T}{\varepsilon}, \sqrt{\frac{L \max(\ln(T/L), 1)}{\varepsilon}}\right)\right)$$
$$= \Omega\left(\min\left(d, K, \frac{T}{\varepsilon}, \sqrt{\frac{K \max(\ln(T/K), 1)}{\varepsilon}}\right)\right).$$

2. *given $L \in \mathbb{N}$ with $8 \leq L \leq T$, there exists an input stream $x \in \mathscr{D}_{\text{likes}}$*

with flippancy K and

$$\frac{L}{16} \leq K \leq \min\left\{L, \frac{T}{4}\right\},$$

and with $\|x^t\|_1 = 1$ for all t (i.e., each update modifies at most one item) such that any ε-differentially private algorithm to the CountDistinct *problem with item-level privacy with error at most α at all time steps with probability at least $2/3$ must satisfy*

$$\alpha = \Omega\left(\min\left(d, K, \sqrt{\frac{K \ln(T/K)}{\varepsilon}}\right)\right).$$

Proof. Assume there is an ε-dp algorithm $\mathscr{A}$ for CountDistinct with error at most α at all time steps with probability at least $2/3$. If $\alpha > d/2$, then the error is $\Omega(d)$. Also, if $\alpha > L/8$, then $\alpha = \Omega(L)$. Thus, in the following, we consider the case $\alpha \leq \min\{d/2, L/8\}$.

This takes care of the first two terms in the lower bound.

Defining $m = \lfloor 2\alpha \rfloor$, it follows that $m \leq \min(d, L/4)$. We first prove the singleton update lower bound and then prove the one for multiple updates.

Singleton Updates. We first find $T' \leq T$ and $L' \leq L$ such that $4m$ divides T' and m divides L'. If this is not the case for T and L, then pick parameters T' and L' such that

1. $4m$ divides T' and m divides L',
2. $\Delta = T - T' \leq 4m < L/2 \leq T/2$ (i.e., $T' = \Theta(T)$), and
3. $0 \leq L - \Delta - L' \leq m$.

This implies that

$$L' \geq \frac{7L}{8} - \Delta \geq \frac{3L}{8}. \qquad \text{(since } \Delta \leq L/2\text{)}$$

Thus, as $L \leq T$, we get

$$\begin{aligned} 0 &\leq L - \Delta - L' \\ &= L - (T - T') - L' && \text{(since } \Delta = T - T'\text{)} \\ &= T' - L' - (T - L) \\ &\leq T' - L', && \text{(since } L \leq T\text{)} \end{aligned}$$

giving that $L' \leq T'$.

We use T' and L' in the proof below to construct a sequence of length T' fulfilling the statements of the theorem. To complete the proof of the theorem, we append to the sequence $T - T'$ many all-zero vectors to guarantee that the stream has length T.

Appending "blank" operations to the sequence will not invalidate the statements of the theorem.

We now construct a set of input sequences of length T' with flippancy $K := \min(L', T'/4)$ and use them to prove a lower bound for α of

$$\Omega\left(\min\left(K \ln(T'/K), \sqrt{\frac{K \ln(T'/K)}{\varepsilon}}\right)\right).$$

Combined with the above case distinctions giving lower bounds on α of $\Omega(d)$, and $\Omega(L)$, the fact that $K = \Theta(L)$ and that $T' = \Theta(T)$, this implies that the additive error satisfies

$$\alpha = \Omega\left(\min\left(d, K, \sqrt{\frac{K \ln(T/K)}{\varepsilon} + 1}\right)\right).$$

Partition the timeline into T'/m blocks of length m, namely

$$B_1 = [1, m], \quad B_2 = [m+1, 2m], \quad \dots$$

Now, for any subset of blocks $J = (j_1, \dots, j_k)$ with $k := \min(L', T'/4)/m$ and $1 \le j_1 < j_2 < \cdots < j_k \le T'/m$, define an input sequence $x(J)$ such that for any item $i \in [m]$ we insert element i in the i-th time step of every odd block of J, and delete it again at the i-th position of every even block of J. Formally, for any item $i \in [m]$, set

$$x(J)_i^t = \begin{cases} 1 & \text{for all } t = (j_{2p-1} - 1)m + i \in B_{j_{2p-1}},\ p = 1 \dots, \lceil k/2 \rceil \\ -1 & \text{for all } t = (j_{2p} - 1)m + i \in B_{j_{2p}},\ p = 1 \dots, \lceil k/2 \rceil \\ 0 & \text{otherwise} \end{cases}$$

Thus, for every $i \in [m]$, we have

$$f^t(x_i) = \begin{cases} 1 & \text{for all } t \in [(j_{2p-1} - 1)m + i, (j_{2p} - 1)m + i),\ p = 1, \dots, \lceil k/2 \rceil \\ 0 & \text{otherwise} \end{cases}$$

Note that for all items i with $m < i \le d$, we have $f^t(x_i) = 0$ for all $t \in [T']$. These items i with $i > m$, if they exist, are never inserted or deleted.

In total, there are k updates per item $i \in [m]$, and thus exactly K updates in total. Hence, the total flippancy is $K = mk = \min(L', T'/4)$. If $K = L'$, then $L \ge K \ge 3L/8$. If $K = T'/4$, then $L' \le T'$ implies that

$$L \ge L' \ge K = T'/4 \ge L'/4 \ge 3L/32 \ge L/16.$$

Thus in either case $K = \Theta(L')$. Furthermore $K \le T'/4 \le T/4$.

Now let E_J be the set of output sequences where the output of $\mathscr{A}$ is

1. a value of $m/2$ or larger for all time steps $t \in [j_{2p-1}m, (j_{2p} - 1)m]$ with $1 \le p \le \lceil k/2 \rceil$, and
2. smaller than $m/2$ for all time steps t such that
   a) $t < j_1 m$, or
   b) $t \in [j_{2p}m, (j_{2p+1} - 1)m]$ for some $0 \le p < \lceil k/2 \rceil$, or
   c) $t \ge j_k m$.

Note that the set of output sequences E_J for distinct J are disjoint, since for each multiple of m (i.e., the end of a block), it is clearly defined whether the output is at least $m/2$ or smaller than $m/2$, and as such the values $j_1, \dots, j_k$ can be uniquely recovered. Thus, there are $\binom{T'/m}{k}$ disjoint events E_J.

Note that for an input sequence $x(J)$, every output sequence where $\mathscr{A}$ has additive error smaller than $\alpha = m/2$ must belong to E_J. Since the algorithm is correct with probability at least $2/3$, we have that

$$\Pr[\mathscr{A}(x(J)) \in E_J] \ge \frac{2}{3}.$$

Recall that for item-level differential privacy, two input sequences are neighboring if they differ in the data of at most one item. Since two input sequences $x(I)$ and $x(J)$ corresponding to two subsets $I \ne J$ differ in the data of at most m items, it follows by group privacy that

$$\Pr[\mathscr{A}(x(J)) \in E_I] \ge e^{-m\varepsilon} \cdot \frac{2}{3} \tag{8.1}$$

for any two subset of blocks I and J.

Since the output sequences E_J are disjoint for different J, the probability that the algorithm with input $x(I)$ outputs an event E_J for some J is at most 1. More formally, we have:

$$\begin{aligned}
1 &\geq \Pr[\mathscr{A}(x(I)) \in E(J) \text{ for some } J] \\
&= \sum_{J=(j_1,\dots,j_k)} \Pr[\mathscr{A}(x(I)) \in E(J)] && \text{(since the events are disjoint)} \\
&\geq \binom{T'/m}{k} \cdot \Pr[\mathscr{A}(x(I)) \in E(J)] && \text{(by the bound on number of such } J\text{s)} \\
&\geq \binom{T'/m}{k} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} && \text{(by Equation (8.1))} \\
&\geq \frac{(T'/m)^k}{(k)^k} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} \\
&= \frac{T'^{(K/m)}}{K^{K/m}} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} && \text{(since } k = K/m\text{)}
\end{aligned}$$

This gives

$$m^2 + \frac{m \ln(3/2)}{\varepsilon} \geq \frac{K \ln(T'/K)}{\varepsilon}$$

which implies

$$m = \Omega\left(\min\left(K \ln(T'/K), \sqrt{\frac{K \ln(T'/K)}{\varepsilon}}\right)\right).$$

Note that since $T' \geq 4K$, $\ln(T'/K) \geq \ln(4) > 1$. This completes the proof.

Multiple Updates. We first find $T' \leq T$ and $L' \leq L$ such that 4 divides T' and m divides L'. If this is not the case for T and L, then pick parameters T' and L' such that

The proof broadly follows the same structure as the above one, but is simpler, since we can update multiple items simultaneously.

1. 4 divides T' and m divides L',
2. $\Delta = T - T' \leq 4$ (i.e. $T' = \Theta(T)$), and
3. $\Delta m \leq L - L' \leq (\Delta + 1)m$.

This implies that $L' \geq L - (\Delta + 1)m \geq L - 5m \geq 3L/8$. We use T' and L' in the proof below to construct a sequence of length T' fulfilling the statements of the theorem. To complete the proof of the theorem, we append to the sequence $T - T'$ many all-zero vectors to guarantee that the stream has length T.

As earlier, appending sequence "blank" operations will not invalidate the statements of the theorem.

The idea is similar to above, only we do not define blocks, but directly choose $k := \min(L'/m, T'/4)$ time steps in which all items in $[m]$ are updated. Thus the flippancy K will equal mk. More precisely, we construct the following set of input sequences. For any $I = (t_1, \dots, t_k)$ with $1 \leq t_1 < t_2 < \cdots < t_k \leq T'$, we define an input sequence $x(I)$ as follows: For any item $i \in [m]$, set

$$x(I)_i^t = \begin{cases} 1 & \text{for all } t = t_j \text{ with } j \text{ odd} \\ -1 & \text{for all } t = t_j \text{ with } j \text{ even} \\ 0 & \text{otherwise} \end{cases}$$

In total, there are k updates per item in $[m]$, thus, exactly K updates in total, i.e., the total flippancy equals $K = \min(L', mT'/4)$. This implies that $\min(3L/8, T/4 - 1) \leq K \leq \min(L, dT/4)$.

As earlier, for $I = (t_1, \dots, t_k)$ with $1 \leq t_1 < t_2 < \cdots < t_k \leq T'$, let E_I be the set of output sequences with

1. a value of $m/2$ or larger at all time steps $t \in [t_{2p-1}, t_{2p})$ for some $1 \leq p \leq \lceil k/2 \rceil$, and
2. a value smaller than $m/2$ at all time steps t where
   a) $t \leq t_1$, or
   b) $t \in [t_{2p}, t_{2p+1})$ for some $0 \leq p < \lceil k/2 \rceil$.

Note that the events E_I and E_J for any $I \neq J$ are disjoint, since in the event E_I it is clearly defined for every time step whether the output is at least $m/2$ or smaller than $m/2$, and from that the set I can be uniquely recovered. Thus, there are $\binom{T'}{k}$ disjoint events E_J.

Note that for input sequence $x(I)$ every output sequence where $\mathscr{A}$ has an additive error smaller than $m/2$ must be in E_I. As the algorithm is correct with probability at least 2/3,

$$\Pr[\mathscr{A}(x(I)) \in E_I] \geq \frac{2}{3}.$$

As two input sequences $x(I)$ and $x(J)$ with $I \neq J$ differ in the data of at most m items, it follows by group privacy that

$$\Pr[\mathscr{A}(x(I)) \in E_J] \geq e^{-m\varepsilon} \cdot \frac{2}{3}. \tag{8.2}$$

The probability that with input $x(I)$ the algorithm outputs an E_J for some j is at most 1. Thus we have

$$\begin{aligned}
1 &\geq \Pr[\mathscr{A}(x(I)) \in E(J) \text{ for some } J] \\
&= \sum_{J=(j_1,\ldots,j_k)} \Pr[\mathscr{A}(x(I)) \in E(J)] && \text{(since the events are disjoint)} \\
&\geq \binom{T'}{k} \cdot \Pr[\mathscr{A}(x(I)) \in E(J)] && \text{(by the bound on number of such } J\text{s)} \\
&\geq \binom{T'}{k} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} && \text{(by Equation (8.2))} \\
&\geq \frac{T'^k}{(k)^k} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} \\
&= \frac{T'^{(K/m)}}{K^{K/m}} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} && \text{(since } k = K/m\text{)}
\end{aligned}$$

as earlier. Next we consider two cases, the first one resulting in two different lower bounds on m and the second one giving a third lower bound on m. The combination of these three lower bounds then gives the claimed bound above of

$$\alpha = \frac{m}{2} = \Omega\left(\min\left(\frac{T'}{\varepsilon}, \sqrt{\frac{K \max\left(\ln(T'/K), 1\right)}{\varepsilon}}, K \max\left(\ln\left(T'/K\right), 1\right)\right)\right)$$

Case 1: $L' < mT'/4$. In this case $K = L'$ and we have

$$m^2\varepsilon + m\ln(3/2) \geq K \ln(T'm/K) \geq K \max(\ln(T'/K), 1)$$

where the last inequality holds since $K \leq mT'/4$, i.e., $\ln(T'm/K) \geq \ln(4) > 1$. Hence

$$m = \Omega\left(\min\left(\sqrt{\frac{K \max(\ln(T'/K), 1)}{\varepsilon}}, K \max(\ln(T'/K), 1)\right)\right).$$

As $K = L' = \Theta(L)$, the same bound with K replaced by L follows.

Case 2: $L' \geq mT'/4$. This implies that $K = mT'/4$ and, thus, that there are updates in at least $T'/4$ many time steps. In this case, the inequality derived above can be reformulated as:

$$\begin{aligned} 1 &\geq \frac{T'^{K/m}}{(K/m)^{K/m}} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} \\ &= 4^{T'/4} \cdot e^{-m\varepsilon} \cdot \frac{2}{3} && \text{(since } K/m = T'/4\text{)} \\ &= e^{T' \ln 4/4 - m\varepsilon} \cdot \frac{2}{3} \end{aligned}$$

which implies that the inequality leads to a contradiction for $m = \Omega(T'/\varepsilon)$.

These two cases show that

$$\alpha = \Omega\left(\min\left(\frac{T'}{\varepsilon}, \sqrt{\frac{L\max(\ln(T'/L), 1)}{\varepsilon}}, L\max(\ln(T'/L), 1)\right)\right)$$

for the above input sequence. Combined with the above lower bounds on α of $\Omega(\min(d, L))$ and since $T' = \Theta(T)$, the claimed lower bound follows. □

8.7. Lower Bounds for Approximate Differential Privacy

In this section, we adapt the lower bounds by Jain et al. [19] for item-level differential privacy to our parameter scheme.

Theorem 8.7.1 *Given $\varepsilon, \delta \in (0, 1]$,*

1. *let K and T be sufficiently large parameters. There exists a dimension $d \in \mathbb{N}$ and an input stream $x \in \mathscr{D}_{\text{likes}}$ with flippancy at most K such that any item-level, (ε, δ)-differentially private algorithm for the CountDistinct problem with error at most α at all time steps with probability at least 0.99 must satisfy*

$$\alpha = \Omega\left(\min\left(\frac{\sqrt{T}}{\varepsilon \log T}, \frac{(K\varepsilon)^{1/3}}{\varepsilon \log(K\varepsilon)}\right)\right)$$

2. *let K and T be sufficiently large parameters satisfying $K \leq T$. There exists a dimension $d \in \mathbb{N}$ and an input stream $x \in \mathscr{D}_{\text{likes}}$ with flippancy at most K and satisfies $\|x^t\|_1 = 1$ for all t, such that any item-level, (ε, δ)-differentially private algorithm to the CountDistinct problem with error at most α at all time steps with probability at least 0.99 must satisfy*

$$\alpha = \Omega\left(\frac{K^{1/3}}{\varepsilon \log K}\right).$$

The reduction by Jain et al. [19] is based on a lower bound for the 1-way marginals problem. Here, the database y is an table consisting of n rows and m columns, where every entry is in $\{0, 1\}$. Two databases y and y' are neighbouring if they differ in at most one row. The goal is to estimate the average column sums, i.e., the vector $(\sum_{i=1}^{n} y[i, j])_{j \in [m]}$. The following lower bound [149] holds for estimating 1-way marginals under approximate dp.

[149]: Bun et al. (2018), "Fingerprinting Codes and the Price of Approximate Differential Privacy"

Lemma 8.7.2 (Bun, Ullman, and Vadhan [149]) *Given $\varepsilon \in (0,1]$, $\delta = o(1/n)$, $\gamma \in (0,1)$, and $m, n \in \mathbb{N}$, any algorithm for 1-WM which is (ε,δ)-differentially private and has error at most γ with probability 0.99 satisfies*

$$n = \Omega\left(\frac{\sqrt{m}}{\gamma\varepsilon \log m}\right).$$

Algorithm 8.2: Algorithm 5 by Jain et al. [19].

Input: Database $y \in \{0,1\}^{n\times m}$, and black-box access to a mechanism M for CountDistinct.
Output: Estimates of marginals $b = (b[1], \dots, b[m])$

for $j = 1, \dots, m$ **do**
  for $i = 1, \dots, n$ **do**
    Set $z^{(j)}[i] \leftarrow e_i$
    Set $z^{(j)}[i+n] \leftarrow -e_i$
Run M on $x \mapsto z^{(1)} \circ z^{(2)} \circ \dots \circ z^{(m)}$ and record answer vector r
for $j \in [m]$ **do**
  $b[j] \leftarrow r[(2j-1)n]/n$
output b

10: So each row in the 1-way marginals problem gives an item in the CountDistinct problem.

Algorithm 8.3: Reduction from 1-WM to CountDistinct for arbitrarily many updates.

Input: Database $y \in \{0,1\}^{n\times m}$ and black-box access to a mechanism M for CountDistinct.
Output: Estimates of marginals $b = (b[1], \dots, b[m])$

for $j = 1, \dots, m$ **do**
  Set $z^{(j)}[1] \leftarrow y^T[j]$
  Set $z^{(j)}[2] \leftarrow -y^T[j]$
Run M on $x \mapsto z^{(1)} \circ z^{(2)} \circ \dots \circ z^{(m)}$ and record answer vector r
for $j \in [m]$ **do**
  $b[j] \leftarrow r[(2j-1)]/n$
output b

Proof Sketch of Theorem 8.7.1. We start by arguing about Item 2. For this case, our example stream is exactly the same as given by Jain et al. [19] in Algorithm 5 (reproduced here as Algorithm 8.2).

They give a reduction from the 1-way marginals problem: For any instance $\mathcal{I}$ of the 1-way marginals problem with n rows and m columns, there is an instance $C(\mathcal{I})$ of CountDistinct with $T = 2mn$, such that if $\mathcal{I}$ and $\mathcal{I}'$ are neighbouring, then $C(\mathcal{I})$ and $C(\mathcal{I}')$ are item-neighbouring. Further, if we can solve $C(\mathcal{I})$ within error α, we can solve $\mathcal{I}$ within error α/n. It follows by Lemma 8.7.2 that

$$\alpha = \Omega\left(\min\left(\frac{\sqrt{m}}{\varepsilon \log m}, n\right)\right).$$

In the instance they constructed, $d = n$.[10] Further, the total flippancy K can be as large as $2mn$ for worst case inputs. Thus, in order to apply the reduction, we need $2mn \le K \le T$. Given parameters $K \le T$, we choose $m = K/(2n)$. The lower bound $\Omega\left(\min\left(\frac{\sqrt{m}}{\varepsilon \log m}, n\right)\right)$ translates to

$$\Omega\left(\min\left(\frac{\sqrt{K/(2n)}}{\varepsilon \cdot \log(K/(2n))}, n\right)\right).$$

For $n = \frac{K^{1/3}}{2(\varepsilon \log K)^{2/3}}$, we have

$$\frac{\sqrt{K/(2n)}}{\varepsilon \log(K/(2n))} \ge \frac{K^{1/3}\varepsilon^{1/3}\log^{1/3} K}{\varepsilon \log(K^{1/2})} = \Omega\left(\frac{K^{1/3}\log^{1/3} K}{\varepsilon^{2/3}\log K}\right) = \Omega(n).$$

Thus, we get

$$\alpha = \Omega\left(\frac{K^{1/3}\log^{1/3} K}{\varepsilon^{2/3}\log K}\right).$$

For Item 1, where we allow general updates, we have to slightly modify the example from Jain et al. [19]: namely, in their Algorithm 5, we collapse every one of their vectors $z^{(j)}$, $j = 1, \dots, m$, into vectors of length 2, one time step for all insertions corresponding to column j, and one time step for all deletions corresponding to column j.

We summarize this in Algorithm 8.3. We then get a reduction with the same properties as before, except that $T = 2n$ and K can be as large as $2mn$. Now, the analysis from Jain et al. [19] can be repeated with our T taking the role of w_x in their analysis, and our K taking the role of T in their analysis. □

8.8. Unknown Total Flippancy

The algorithms from Section 8.4 can be easily extended to the case where the total flippancy K is not known beforehand, at the cost of polylog(K) factors

in the error bound, using Algorithm 8.4. The algorithm repeatedly "guesses" K and then runs the algorithm from earlier with the current guess.

Lemma 8.8.1 *Given $\varepsilon, \delta \in [0, 1)$, Algorithm 8.4 is (ε, δ)-differentially private.*

Proof. By Lemma 8.4.2, the j-th instance of Algorithm 8.1 is $(\varepsilon_j, \delta_j)$-dp. Since $\sum_{j=1}^{\infty} \varepsilon_j = \varepsilon$ and $\sum_{j=1}^{\infty} \delta_j = \delta$, by Fact 3.2.5, Algorithm 8.4 is (ε, δ)-dp. □

Lemma 8.8.2 *For $\delta = 0$, the error of Algorithm 8.4 is at most*

$$O\left(\ln K \sqrt{\frac{K \ln(T \ln K/\beta)}{\varepsilon}} + \frac{\ln^2 K \ln(T \ln K/\beta)}{\varepsilon}\right).$$

For $\delta > 0$, the error of Algorithm 8.4 is at most

$$O\left(\left(\frac{K \ln^2 K \ln(\ln K/\delta) \ln^2(T \ln K/\beta)}{\varepsilon}\right)^{1/3} + \frac{\ln^2 K \sqrt{\ln(\ln K/\delta)} \ln(T \ln K/\beta)}{\varepsilon}\right).$$

Proof. Let j_l be the value of variable j at the end of the stream. For any $j < j_l$, note that by Lemma 8.4.3, with probability at least $1 - \beta_j$, by the choice of S_{K_j}, the algorithm does not abort before having seen the entire stream if the total flippancy is at most K_j. Thus, when the algorithm aborts for some $j < j_l$, we know that the flippancy is at least K_j, and the bound from Lemma 8.4.3 holds for the j-th instance of Algorithm 8.1 with S_{K_j}.

Since the algorithm aborts for all $j < j_l$, we can conclude that the total flippancy of the stream processed by the j-th run of Algorithm 8.1 is at least K_j. Since $\sum_j \beta_j = \beta$, with probability at least $1 - \beta$, we have that

- the total flippancy K is at least $\sum_{j<j_l} K_j = 2^{j_l} - 1$, and
- the bound from Lemma 8.4.3 holds for all instances of Algorithm 8.1 (with their respective parameters).

It then follows that

- $K \geq K_{j_l} - 1 \geq K_j$ for all $j < j_l$, and
- $j_l = O(\ln K)$.

The maximum error over the stream is the maximum error of any instance of Algorithm 8.1. Since $K_j = O(K)$, $\varepsilon_{j_l} \leq \varepsilon_j$ and $\delta_{j_l} \leq \delta_j$ for all $j \leq j_l$, the final bound is now obtained by plugging K, $\varepsilon_{j_l} = \Theta(\varepsilon/j^2)$ for ε, $\delta_{j_l} = \Theta(\delta/j^2)$ for δ, and $\beta_{j_l} = \Theta(\beta/j^2)$ for β into the bound from Lemma 8.4.3, and upper bounding j^2 by $\log^2 K$. □

One can also obtain the minimum of the bound from Lemma 8.8.2 and $\min(K, T, d)$ at the cost of an additive $\varepsilon \ln^2 K \ln(\ln K/\beta)$ factor with a slightly more involved algorithm, by choosing either to not update the output or to abort if there is a trivial algorithm which performs better for the current estimate of K. If we knew the value of K beforehand, we could choose the best algorithm upfront. Not knowing the value of K makes the algorithm slightly more complicated. In the following, we show how to obtain this bound. The full algorithm is given in Algorithm 8.5.

Algorithm 8.4: CountDistinct when K is unknown

Input: Database $x^1, x^2, \ldots$, initial counts $c_1, \ldots, c_d$ (default 0), parameters ε, δ, β, stream length bound T

Output: CountDistinct

```
1  t ← 1 and K_1 ← 2
2  for j = 1, ... do
3      ε_j ← 6ε/(π²j²)
4      δ_j ← 6δ/(π²j²)
5      β_j ← 6β/(π²j²)
6      if δ = 0 then
7          S_{K_j} ← sqrt(K_j ε_j / (18 ln(T/β_j))) + 1
8      else
9          S_{K_j} ← (K_j ε_j / (36 sqrt(ln(1/δ_j) ln(T/β_j))))^{2/3} + 1
10     Run Algorithm 8.1 on x^t, x^{t+1}, ..., with inputs
       c_1, ..., c_d, ε_j, δ_j, β_j, T, S_{K_j} until it aborts
11     Let t' be the last time step processed by Algorithm 8.1
12     t ← t' + 1, j ← j + 1
13     K_j ← 2^j
```

Algorithm 8.5: CountDistinct, unknown K, all bounds

Input: Database $D = x^1, x^2, \dots$, initial counts $c_1, \dots, c_d$ (default 0), parameters ε, δ, β, stream length bound T.

Output: Private counts under CountDistinct with unknown K.

$t \leftarrow 1$ and $K_1 \leftarrow 2$
for $j = 1, \dots$ **do**
  $\varepsilon_j \leftarrow 12\varepsilon/(\pi^2 j^2)$, $\delta_j \leftarrow 6\delta/(\pi^2 j^2)$, and $\beta_j \leftarrow 12\beta/(\pi^2 j^2)$
  if $\delta = 0$ **then**
    $S_{K_j} \leftarrow \sqrt{K_j \varepsilon_j / (18 \ln(T/\beta_j))} + 1$
    $B_j \leftarrow \sqrt{\varepsilon^{-1} K \ln(T/\beta)}$
    $\mathrm{err}_T \leftarrow \varepsilon^{-1} T \log(T/\beta)$
  else
    $S_{K_j} \leftarrow \left(\frac{K_j \varepsilon_j}{36\sqrt{\ln(1/\delta_j)\ln(T/\beta_j)}} \right)^{2/3} + 1$
    $B_j \leftarrow \left(\varepsilon_j^{-2} K_j \ln(1/\delta_j) \ln^2(T/\beta_j)\right)^{1/3} + \varepsilon_j^{-1}\sqrt{\ln(1/\delta_j)}\ln(T/\beta_j)$
    $\mathrm{err}_T \leftarrow \varepsilon^{-1}\sqrt{T \ln(1/\delta) \log(T/\beta)}$
  if $\min(K_j, B_j) > \min(d, \mathrm{err}_T)$ **then**
    if $\min(d, \mathrm{err}_T) = d$ **then**
      Abort and switch to the trivial algorithm that always outputs 0
    else
      if $\delta = 0$ **then**
        Abort and switch to the Laplace mechanism with $\Delta_1 = T$
      else
        Abort and switch to the Gaussian mechanism with $\Delta_2 = \sqrt{T}$
  else if $K_j \geq B_j$ **then**
    Run Algorithm 8.1 on $x^t, x^{t+1}, \dots$ with inputs $c_1, \dots, c_d, \varepsilon_j, \delta_j, \beta_j, T, S_{K_j}$ until it aborts
  else
    Run Algorithm 8.1* (the same as Algorithm 8.1, without updating out) on $x^t, x^{t+1}, \dots$ with inputs $c_1, \dots, c_d, \varepsilon_j, \delta_j, \beta_j, T, S_{K_j}$ until it aborts.
  Let t' be the last time step processed by the monitoring mechanism
  out $\leftarrow \sum_{i=1}^{d} f^{t'}(x_i) + \mathrm{Lap}(1/\varepsilon_j)$
  $t \leftarrow t' + 1$, $j \leftarrow j + 1$, and $K_j \leftarrow 2^j$

Theorem 8.8.3 *Given $d, T \in \mathbb{N}$, $\varepsilon, \delta, \beta > 0$, and letting K be the actual flippancy of the input stream that is unknown to the algorithm, there exists*

1. *an item-level ε-differentially private algorithm for the CountDistinct problem in the general model with error at most*

$$O\left(\min\left(d,\ K,\ \ln K\sqrt{\frac{K\ln(T/\beta)}{\varepsilon}},\ \frac{T\log(T/\beta)}{\varepsilon}\right) + \frac{\ln^2 K \ln(\ln K/\beta)}{\varepsilon}\right)$$

   at all time steps with probability $1-\beta$, and
2. *an item-level (ε, δ)-differentially private algorithm for the CountDistinct problem in the general model with error at most*

$$O\left(\min\left(d,\ K,\ \left(\frac{K\ln^2 K \ln(1/\delta)\ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3},\ \frac{\sqrt{T\ln(1/\delta)\log(T/\beta)}}{\varepsilon} + \frac{\ln^2 K \ln(\ln K/\beta)}{\varepsilon}\right)\right)$$

   at all time steps with probability at least $1-\beta$, for any $\varepsilon < 1$.

Proof. Since Algorithm 8.5 is a composition of Algorithm 8.4 with parameters $\varepsilon/2$ and δ and a sequence of Laplace mechanisms such that the j-th Laplace mechanism is ε_j-dp, with $\sum_{j=1}^{\infty} \varepsilon_j = \varepsilon/2$, privacy follows.

For accuracy, first assume that we do not abort before we have seen the entire stream. The parameters t, j and K_j are exactly the same as in Algorithm 8.4. Thus, we have $j \leq \log^2 K$ and $K_j = O(K)$ for all j used in the algorithm. We condition on the accuracy bound from Lemma 8.4.3 holding for all runs of Algorithm 8.4, and the Laplace noise $\mu_j \sim \text{Lap}(1/\varepsilon_j)$ satisfying $|\mu_j| \leq \varepsilon_j^{-1} \log(1/\beta_j)\ \forall j$. By Lemma 8.4.3, the accuracy bound is true with probability $\geq 1 - \beta/2$. By Fact 3.2.2, the bound on μ holds with probability $1 - \sum_j \beta_j \geq 1 - \beta/2$. Both bounds hold together with probability $1 - \beta$.

We first condition on the random variables being well-behaved.

Now, consider the case where $\delta = 0$. For any j such that $K_j > B_j$, we have that the error of the run of Algorithm 8.1 in the j-th round is bounded by

$$O\left(\sqrt{\frac{K_j \ln(T/\beta_j)}{\varepsilon_j}} + \frac{\ln(T/\beta_j)}{\varepsilon_j}\right)$$

by the conditioning above. We have that

$$K_j > B_j \implies \frac{\ln(T/\beta_j)}{\varepsilon_j} < \sqrt{\frac{K_j \ln(T/\beta_j)}{\varepsilon_j}}, \qquad \text{(by multiplying with } \sqrt{\ln(T/\beta_j)/K_j\varepsilon_j}\text{)}$$

Thus, for any j such that $K_j > B_j$, the error is bounded by

$$\begin{aligned} O\left(\sqrt{\frac{K_j \ln(T/\beta_j)}{\varepsilon_j}}\right) &= O\left(\min\left(K_j, \sqrt{\frac{K_j \ln(T/\beta_j)}{\varepsilon_j}}\right)\right) \\ &= O\left(\min\left(K, \ln K \sqrt{\frac{K \ln(T \ln K/\beta)}{\varepsilon}}\right)\right). \end{aligned}$$

For any j such that $K_j \leq B_j$, we do not update the output until the end of round j. Let t_{j-1} be the time step where the $(j-1)$-st copy of Algorithm 8.1 or Algorithm 8.1* aborted. The flippancy bound for all $t \in [t_{j-1}, t_j)$ is

$$\left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \sum_{i=1}^{d} f^{t}(x_i)\right| \leq K_j. \qquad \text{(by Lemma 8.4.3)}$$

Since the output at all those time steps is given by $\text{out} = \sum_{i=1}^{d} f^{t_{j-1}}(x_i) + \mu_j$,

$$\begin{aligned} \left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \text{out}\right| &= \left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \sum_{i=1}^{d} f^{t}(x_i) - \mu_j\right| \\ &\leq \left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \sum_{i=1}^{d} f^{t}(x_i)\right| + |\mu_j| && \text{(by triangle inequality)} \\ &\leq K_j + |\mu_j| && \text{(by the flippancy bound)} \\ &\leq K_j + \frac{\log(1/\beta_j)}{\varepsilon_j} && \text{(by the conditioning on } \mu_j\text{)} \\ &= O\left(\min\left(K_j, \sqrt{\frac{K_j \ln\left(T/\beta_j\right)}{\varepsilon_j}}\right) + \frac{\log\left(1/\beta_j\right)}{\varepsilon_j}\right) \\ &= O\left(\min\left(K, \ln K \sqrt{\frac{K \ln(T \ln K/\beta)}{\varepsilon}}\right) + \frac{\ln^2 K \ln\left(\ln K/\beta\right)}{\varepsilon}\right). \end{aligned}$$

Next, consider the case where $\delta > 0$. For any j such that $K_j > B_j$, we have that the error of the run of Algorithm 8.1 in the j-th round is bounded by

$$O\left(\left(\frac{K_j \ln(1/\delta_j)\ln^2(T/\beta_j)}{\varepsilon_j^2}\right)^{1/3} + \frac{\sqrt{\ln(1/\delta_j)}\ln(T/\beta_j)}{\varepsilon_j}\right)$$

by our conditioning. Note that

$$\frac{\sqrt{\ln(1/\delta_j)}\ln(T/\beta_j)}{\varepsilon_j} > \frac{K_j \ln(1/\delta_j)\ln^2(T/\beta_j)}{\varepsilon_j^2} \implies \frac{\sqrt{\ln(1/\delta_j)}\ln(T/\beta_j)}{\varepsilon_j} > K_j,$$

in contradiction to $K_j > B_j$. Thus, for any j such that $K_j > B_j$, the error is

$$\begin{aligned}
&= O\left(\left(\frac{K_j \ln(1/\delta_j)\ln^2(T/\beta_j)}{\varepsilon_j^2}\right)^{1/3}\right) \\
&= O\left(\min\left(K_j, \left(\frac{K_j \ln(1/\delta_j)\ln^2(T/\beta_j)}{\varepsilon_j^2}\right)^{1/3}\right)\right) \\
&= O\left(\min\left(K, \left(\frac{K\ln^2 K \ln(\ln K/\delta)\ln^2(T\ln K/\beta)}{\varepsilon}\right)^{1/3}\right)\right).
\end{aligned}$$

For any j such that $K_j \le B_j$, we do not update the output until the end of round j. Let t_{j-1} resp. t_j be the time step where the $(j-1)$st resp. j-th copy of Algorithm 8.1 or Algorithm 8.1* aborted. Note that by Lemma 8.4.3, the flippancy in interval $[t_{j-1}, t_j)$ is at most K_j. Thus, for all $t \in [t_{j-1}, t_j)$,

$$\left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \sum_{i=1}^{d} f^{t'}(x_i)\right| \le K_j$$

Since the output at all those time steps is given by $\text{out} = \sum_{i=1}^{d} f^{t_{j-1}}(x_i) + \mu_j$, a similar argument as earlier gives

$$\left|\sum_{i=1}^{d} f^{t_{j-1}}(x_i) - \text{out}\right| \le K_j + \frac{\log(1/\beta_j)}{\varepsilon_j}.$$

Lastly, we argue about what happens if we abort and switch to one of the trivial algorithms. This happens exactly when we reach a j such that $\min(K_j, B_j) > \min(d, \text{err}_T)$. Note that up to $j-1$, by the analysis before, we have that the error is bounded by

$$\min(K_{j-1}, B_{j-1}) + \frac{\log(1/\beta_{j-1})}{\varepsilon_{j-1}} \le \min(d, \text{err}_T) + \frac{\log(1/\beta_{j-1})}{\varepsilon_{j-1}}$$

since the algorithm did not abort. After we abort, the error of the algorithm is $O(\min(d, \text{err}_T))$. Thus, the algorithm at any time step is bounded by

$$O\left(\min(d, \text{err}_T) + \frac{\ln^2 K \log(1/\beta)}{\varepsilon}\right).$$

Since we have that $K_j \le K$, we have for $\delta = 0$ that

$$\min(d, \text{err}_T) \le \min\left(K,\ \ln K\sqrt{\frac{K\ln(T\ln K/\beta)}{\varepsilon}}\right),$$

and for $\delta > 0$ that

$$\min(d, \mathrm{err}_T) \leq \min\left(K, \left(\frac{K \ln^2 K \ln(1/\delta) \ln^2(T/\beta)}{\varepsilon^2}\right)^{1/3}\right). \qquad \square$$

9. Private k-core decomposition

The urban environment of Madras was too densely populated, too diverse, too large, and too interconnected with its surroundings for there to be an effective way to contain cholera once it had broken out.

Michael Zeheter, *Epidemics, Empire, and Environments*

9.1. Introduction

Core decomposition and densest subgraph are fundamental problems that can be used to identify structure within graphs, with wide-ranging practical applications. Recently, motivated by scenarios where the graph encodes confidential user information, there has been a growing interest in developing "privacy-preserving" algorithms for these problems.

See Dhulipala et al. [20] for some discussion on applications.

Such algorithms have been designed for both the *centralized* and *local* models of differential privacy. In the centralized model, a *trusted* party runs the algorithm, with full access to all user data (here, the entire graph). The local model removes the trust assumption and is more distributed. In this model, each user (here, a vertex in the graph) interactively discloses information about its data (here, its incident edges) in multiple *rounds* of communication with an *untrusted* server[1], which produces the output. All messages sent throughout the interaction, not just the output, must be private.

1: The server may accidentally leak their messages, e.g., due to security breaches. However, it is not malicious. This is referred to as "honest but curious" in the literature.

In both models, a primary metric of interest is the *accuracy* of the output. This is fairly well understood in the centralized model, where there are fast and private (approximation) algorithms for core decomposition [150] and densest subgraph [20, 150–153] with optimal and near-optimal additive error, respectively. However, there are still gaps in our understanding of this metric (and its trade-off with round complexity) in the local model. In a groundbreaking paper, Dhulipala et al. [20] initiated the study of both problems in the local model by showing various trade-offs between approximation ratio, additive error, and round complexity. In particular, they gave a private $(2+\eta)$-approximate algorithm for core decomposition that has $O(\log^3 n)$ additive error and $O(\log n)$ round complexity, as well as a private $(4+\eta)$-approximate algorithm for densest subgraph with the same trade-off.

[20]: Dhulipala et al. (2022), "Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs"

[150]: Dhulipala et al. (2023), "Near-Optimal Differentially Private k-Core Decomposition"

[151]: Nguyen et al. (2021), "Differentially Private Densest Subgraph Detection"

[152]: Farhadi et al. (2022), "Differentially Private Densest Subgraph"

[153]: Dinitz et al. (2025), "Almost Tight Bounds for Differentially Private Densest Subgraph"

More recently, Dinitz et al. [153] gave private exact and $(2+\eta)$-approximate local algorithms for densest subgraph with improved additive error, roughly $O(\log^2 n)$, and with $O(n^2 \log n)$ and $O(\log n)$ round complexity, respectively.

It is not clear, however, whether these are the best possible trade-offs. Thus, we seek to understand the following question:

What is the minimum additive error achievable, with high probability, for differentially private core decomposition (and densest subgraph) on n-vertex graphs, in the local model?

9.1.1. Our Contributions

We focus on the pure dp setting in the introduction.

Lower Bounds. We prove a lower bound on the additive error of approximate core decomposition in the centralized model, which carries over to the (less powerful) local model, regardless of round complexity. We also take the first step towards proving an accuracy and round complexity trade-off for exact core decomposition, by showing that 1-round algorithms must be inaccurate. To the best of our knowledge, these are the first lower bounds for core decomposition in either model. Specifically we show the following lower bounds.

Theorem 9.5.1

Centralized and local lower bounds. For constant $\gamma \geq 1$, we show that any *centralized* algorithm for γ-approximate core decomposition has $\Omega(\gamma^{-1} \log n)$ additive error, with constant probability. Since centralized algorithms can simulate local ones, the bound also holds in the local model, regardless of the round complexity of the algorithm.

Theorem 9.5.2

Single round local lower bounds. We show that any local algorithm for exact core decomposition that communicates for a single round has $\Omega(\sqrt{n})$ additive error, with constant probability, on a large family of graphs.

Upper Bounds. To show that our first lower bound (which is already tight in the centralized model) is unlikely to be improved in the local model, we give local algorithms for exact and $(2+\eta)$-approximate core decomposition with improved, resp. nearly matching error, which exhibit new trade-offs with round complexity. In particular, we show that it is possible to achieve an additive error without a linear dependence on the round complexity, answering a question of Dhulipala et al. [20].

We obtain our results via a simple reduction from continual counting of bits in the centralized model to core decomposition in the local model. In particular, in our algorithms, users employ continual counting mechanisms in a black-box manner. Thus, improving the accuracy of continual counting mechanisms (to match the known lower bound of $\Omega(\log T)$ [22]) would immediately improve the accuracy of our algorithms (to optimal).

[22]: Dwork et al. (2010), "Differential privacy under continual observation"

Using a known reduction from densest subgraph to core decomposition, our results for core decomposition immediately lead to local algorithms for 2 and $(4+\eta)$-approximate densest subgraph with the same trade-offs.

Specifically, we show the following results (see also Table 9.1). We note that all our mechanisms satisfy pure differential privacy.

Theorem 9.4.1

Exact core decomposition. We give a local algorithm for exact core decomposition that has $O(\log n \log \Delta)$ additive error[2] with high probability and $O(n)$ round complexity.

2: Here Δ is the maximum degree of the graph.

Theorem 9.4.5

$(2+\eta)$-approximate core decomposition. For constant $\eta > 0$, we give a local algorithm for $(2+\eta)$-approximate core decomposition that has nearly optimal additive error, $O(\log n \log\log n)$, with high probability, and $O(\log^2 n)$ round complexity.

Theorem 9.4.6

Approximate densest subgraph. For constant $\eta > 0$, we give local algorithms for 2-approximate and $(4+\eta)$-approximate densest subgraph[3] with $O(\log n \log \Delta)$ and $O(\log n \log\log n)$ error, with high probability, and round complexity $O(n)$ and $O(\log^2 n)$, respectively.

3: Here, the algorithm returns a set of vertices, which induce a subgraph whose density is approximately maximum.

Problem	Apx. Factor	Additive Error	Rounds	Ref.
Core Decomp.	1	$O(\log n)$	$O(n)$	[150]
	1	$O(\log n \log \Delta)$	$O(n)$	9.4.1
	1	$\Omega(\sqrt{n})$	1	9.5.2
	$2+\eta$	$O(\log n)$	$O(\log n)$	[150]
	$2+\eta$	$O(\log^3 n)$	$O(\log n)$	[20]
	$2+\eta$	$O(\log n \log\log n)$	$O(\log^2 n)$	9.4.5
	γ	$\Omega(\gamma^{-1} \log n)$	any	9.5.1
Densest Subgraph	1	$O(\log(n\delta^{-1}) \log n)$	$O(n^2 \log n)$	[153]
	1	$\Omega(\sqrt{\log n})$	any	[151]
	2	$O(\log n \log \Delta)$	$O(n)$	9.4.6
	$2+\eta$	$O(\eta^{-1} \log^2 n)$	$O(\eta^{-1} \log n)$	[153]
	$4+\eta$	$O(\log^3 n)$	$O(\log n)$	[20]
	$4+\eta$	$O(\log n \log\log n)$	$O(\log^2 n)$	9.4.6

Table 9.1.: Summary of error bounds in the local model. Each upper bound is for a mechanism satisfying (ε, δ)-edge differential privacy and holds with high probability. If δ appears in the expression, then $0 < \delta < n^{-\Omega(1)}$; otherwise, $\delta = 0$. Some of our error bounds depend on the maximum degree, Δ, of the input graph. Lower bounds are for ε-differential privacy and hold with constant probability. To save space, the dependency on ε is omitted from each additive error upper bound.

9.1.2. Related Work

Dhulipala et al. [150] have concurrently presented a local algorithm for exact core decomposition with additive error $O(\log n)$. Similar to our local algorithm, this algorithm privately implements the classic, exact peeling algorithm. However, they use different techniques (a multi-dimensional variant of the sparse vector technique). The results presented in this paper were combined with their results, and the resulting paper has been published in the proceedings of the European Symposium on Algorithms (ESA) 2025.

Continual counting has been used in the centralized model, along with other techniques, to obtain a $(2+\eta)$-approximate private densest subgraph algorithm with additive error $O(\log^{2.5} n)$, with high probability, which runs in near-linear time [152]. Their privacy analysis has a similar flavor as ours, but is more complex, while the accuracy analysis is quite different. Finally, there are insertions-only and deletions-only graph algorithms under continual observation that have used the idea of tracking the difference sequence of a desired quantity (e.g. degree of a vertex) via continual counting in the centralized model [129].

[152]: Farhadi et al. (2022), "Differentially Private Densest Subgraph"

[129]: Fichtenberger et al. (2021), "Differentially Private Algorithms for Graphs Under Continual Observation"

9.2. Preliminaries

9.2.1. Graphs

Undirected, Unweighted Graphs. We consider undirected, unweighted graphs $G = (V(G), E(G))$, where $V(G)$ is the set of *vertices/nodes* and $E(G) \subseteq V(G) \times V(G)$ is the set of *edges*.

Adjacency and Degree. A vertex u is *adjacent* to a vertex v if $(u, v) \in E(G)$. The degree of v, denoted $\deg(v)$, is the number of vertices adjacent to v.

Subgraphs. H is a subgraph of G if $V(H) \subseteq V(G)$. The *induced* subgraph for a vertex set $U \subseteq V(G)$, denoted by $G[U]$, is the largest subgraph (in terms of number of edges) of G such that $V(H) = U$.

Core Number. The *coreness* or core number of a vertex v in a graph G, denoted $k_G(v)$, is the largest integer k such that there exists a subgraph H of G that contains v such that, for every vertex $u \in V(H)$, the (induced) degree of u in H is at least k, i.e., $\deg_H(u) \geq k$.

k-core. Given $k \in \mathbb{N}$, the *k-core* of the graph G is the subgraph of G induced by the vertices of G with coreness at least k in G, i.e., $G[U]$, where $U = \{v \in V \mid k_G(v) \geq k\}$[4].

4: An alternative, equivalent definition is that the k-core of G is the (unique) largest subgraph of G in which every vertex has degree at least k.

Problem 5 (Core Decomposition) *Given a graph $G = (V, E)$, the core decomposition problem is to return a vector $(k_G(v) : v \in V)$ containing the coreness of each vertex of G (assuming a fixed ordering of the vertices).*

Density. The *density* $\rho(G)$ of a graph G is given by

$$\rho(G) = \frac{|E(G)|}{|V(G)|}.$$

Notice that, for any subset U of vertices of G, if H is a subgraph of G such that $V(H) = U$, then $\rho(H) \leq \rho(G[U])$.

Problem 6 (Densest Subgraph) *Given a graph $G = (V, E)$, the densest subgraph problem is to return a subset of vertices $U \subseteq V(G)$ such that*

$$\frac{|E(G[U])|}{|U|} = \operatorname*{argmax}_{S \subseteq V} \frac{|E(G[S])|}{|S|}$$

For differential privacy, we need the notion of *neighboring graphs*, which we define as follows.

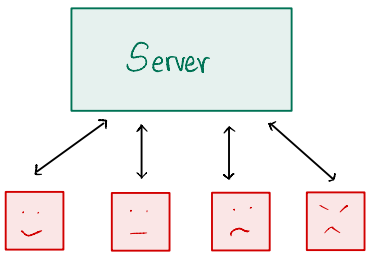
Definition 9.2.1 (Neighboring graphs) *For the database space*

$$\mathscr{D} = \{G \mid G \text{ is a graph on } n \text{ vertices}\},$$

two databases G and G' are neighboring if

$$\text{there exists } e \in V \times V \text{ such that } E(G) \setminus \{e\} = E(G') \setminus \{e\}.$$

9.2.2. Local Differential Privacy



In the *local* model of differential privacy, there are n *users*, each with *private data*, who communicate in synchronous *rounds* with a *server* (or *curator*). In each round, each user can send a message to the server, based on its private data, its local memory, the messages (from the server) that it has received so far, and possibly some *fresh* local randomness. Upon receiving all user messages, the server can either broadcast a message to all users, or end the interaction and produce an output. If it broadcasts a message, then the users will receive it at the end of the round. Subsequently, they may write information to their local memories, which persists into future rounds, before continuing to the next round.

A *local mechanism* $\mathscr{M}$ specifies (randomized) algorithms for the server and for each user, which dictate the messages they send in each round and what they write to their local memories. The *input* of $\mathscr{M}$ is the private data of each user. An *execution* of $\mathscr{M}$ on an input generates a *transcript*, which consists of

the messages sent by the server and all users in every round, if they run the algorithms specified by $\mathscr{M}$, with each user having the corresponding private input data. The *round complexity* of a mechanism is the maximum number of rounds in any transcript. A local mechanism is *ε-differentially private* if it is ε-differentially private, when viewed as a randomized algorithm mapping private user data to transcripts.

When the local model is applied to graphs, there is an underlying *input graph*, G, whose vertices are the users (and public knowledge). The private data of each user (vertex) is its set of neighbors in G. Hence, only users u and v know whether $\{u, v\}$ is an edge in G. In other words, a local mechanism in this setting is *ε-(edge) differentially private* if its transcript is ε-differentially private on neighboring input graphs.

The prevailing formalization of local differential privacy used for graph data [20, 153, 154] was adapted from the general (non-graph) setting [155]. In this formalism, *local randomizers* are used to describe the computations carried out by the users to generate the (differentially private) messages they send in each round. Specifically, an *ε-local randomizer* is a randomized algorithm A that takes as input a subset of vertices of the input graph and returns an output such that, for all subsets of vertices N, N' (corresponding to possible neighborhoods of a user) that differ in at most one element and any subset Y of outputs,

$$\Pr[A(N) \in Y] \leq e^{\varepsilon} \cdot \Pr[A(N') \in Y].$$

[20]: Dhulipala et al. (2022), "Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs"

[153]: Dinitz et al. (2025), "Almost Tight Bounds for Differentially Private Densest Subgraph"

[154]: Eden et al. (2023), "Triangle Counting with Local Edge Differential Privacy"

[155]: Joseph et al. (2019), "The Role of Interactivity in Local Differential Privacy"

Paraphrasing Dhulipala et al. [20], a *local mechanism* is specified by a potentially infinite set of local randomizers, $\mathscr{R}$, together with a function, $\mathscr{A}$, describing how the interaction proceeds. A 0-*round transcript* is the empty sequence. Given a t-round transcript π, $\mathscr{A}(\pi)$ either returns $\perp$, indicating the interaction ends, or a pair $(S_U^{t+1}, S_R^{t+1}, S_\varepsilon^{t+1})$ encoding the set S_U^{t+1} of users who participate in round $t+1$, the set $S_R^{t+1} \subseteq \mathscr{R}$ containing the local randomizer assigned to each participating user, and the corresponding privacy parameters S_ε^{t+1}. Suppose that S_O^{t+1} contains the outputs generated by the local randomizers after each participating user runs its assigned local randomizer on its private data (i.e., neighborhood in the input graph) in round $t+1$. Then the concatenation $\pi \odot (S_U^{t+1}, S_R^{t+1}, S_\varepsilon^{t+1}, S_O^{t+1})$ is a $(t+1)$-*round* transcript. In Section 9.6, we show that local mechanisms in our user/server model can be simulated in the local randomizer formalism. Given this, we prefer to describe our local mechanisms in terms of server and user algorithms, instead of local randomizers.

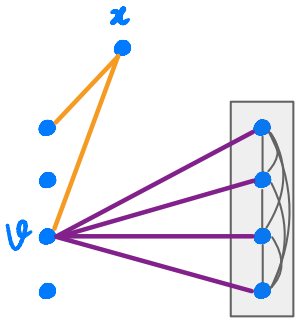
9.3. Technical Overview

Centralized Lower Bound. Our lower bound for γ-approximate core decomposition in the centralized model uses a standard "packing argument" [134]. The idea is to construct a collection of disjoint "bad" output classes for a certain input and show that the algorithm has a "large enough" probability of producing an output in each of the bad classes, due to differential privacy. The only subtlety is that we need to carefully construct the output classes to take advantage of the approximation guarantee.

[134]: Hardt et al. (2010), "On the geometry of differential privacy"

Local Lower Bound. Our lower bound for 1-round, exact core decomposition in the local model is obtained by reduction to a problem in the *centralized* model, for which there is a strong lower bound. Specifically, it is known that, to privately answer $\Theta(n)$ *random* inner product queries on a

[156]: De (2012), "Lower Bounds in Differential Privacy"



secret $\{0,1\}^n$ vector, most responses need to have $\Omega(\sqrt{n})$ additive error [156]. Recently, Eden et al. gave an elegant lower bound on the additive error of 1-round triangle counting in the local model using similar techniques.

In our case, we define a class of "query graphs", one per inner product query, in which the coreness of a *fixed* vertex x (i.e., its assigned "score" in the core decomposition) is (roughly) the answer to the query. In addition to x, there are some *secret* vertices (and other vertices). For each secret vertex v, the existence of the edge $\{v, x\}$ is private (which depends on the secret vector). Crucially, all neighborhoods of x and the secret vertices in *any possible query graph* appear in two specific query graphs, namely, ones corresponding to the all-ones and all-zeros query vectors, respectively. The neighborhoods of the remaining vertices do not depend on the secret vector.

Our approach to solving the inner product problem is now as follows. The centralized algorithm first simulates the 1-round local core decomposition algorithm on the two fixed graphs to determine the messages that x and the secret vertices would send in *any* query graph and saves these messages. Subsequently, when answering a query, the centralized algorithm *reuses* the saved messages for x and the secret vertices (without further privacy loss) when it simulates the core decomposition algorithm on the corresponding query graph. This allows it to answer many queries correctly.

Originally, we had a more complex, direct argument. We briefly mention it here to give an idea of what is going on "under the hood" of the reduction. Specifically, we showed that, on a class of random graphs (similar to the query graphs instantiated in our reduction), most transcripts of a 1-round core decomposition algorithm have the property that, conditioned on seeing the transcript, the coreness of some (fixed) vertex still has high variance, $\Omega(n)$. This implies that the standard deviation of the additive error is $\Omega(\sqrt{n})$. We prefer the simpler reduction argument which also gives a stronger result.

Local Upper Bounds. On the algorithmic side, our starting point is the $(2+\eta)$-approximate core decomposition algorithm of Dhulipala et al. [20], which essentially implements an approximate version of the classic, *exact* peeling algorithm for core decomposition. To remove the approximation factor, it is natural to consider implementing the original algorithm, which is what we do to obtain our local algorithm for exact core decomposition. At a high level, the main technical challenge here is for each user (vertex) to privately disclose its degree as the server continually deletes subsets of users from the graph (based on the disclosed degrees). In the algorithm of Dhulipala et al. [20], each user simply adds fresh Laplace noise to its actual degree and discloses the resulting sum. In this case, the standard deviation of the noise (which dominates the additive error) needs to be *linear in the number of disclosures*, i.e., deletions, to ensure privacy. This is acceptable for the approximate peeling algorithm, where $O(\log^2 n)$ deletions occur, but not for the exact one, where $\Omega(n)$ deletions might occur.

Recall this from Definition 2. We discussed adaptive variants in Chapter 7.

A *continual counting* mechanism privately discloses all (noisy) prefix sums of an input sequence of integers. There are *adaptive* variants, where the elements of the input sequence arrive online, one at a time, and the mechanism discloses the sum after each arrival. Our new approach is for each user to use an adaptive continual counting mechanism to track the *sum of its degree changes* due to deletions. It can then disclose its degree as its (noisy) initial degree plus the last sum.

There are several points that make implementing this non-trivial. First, note that each user runs its own continual counting mechanism whose input (apart from its neighborhood list) is given by the server. Thus, if the inputs

to the mechanisms (on two graphs that differ by a single edge) can differ for each user, even slightly, then the total privacy loss can be $\Omega(n)$, which would require each mechanism to add prohibitively large noise to maintain overall privacy. Through a careful analysis, we show that the total privacy loss is constant. Next, since the outputs of the counting mechanisms have noise, the users do not report their actual degrees, but rather *noisy* ones, so the error could amplify over time. We prove that the exact peeling algorithm is *robust*, in the sense that the error of its output is bounded by the maximum error of the noisy degrees (which is fairly small, with high probability). Finally, in the standard formalism of the local model via local randomizers, users do not have persistent memory. However, it is possible to simulate *any* user/server algorithm (with persistent memory) in the local randomizer model (with computationally unbounded users) (see Lemma 8 by Beimel et al. [157]).

[157]: Beimel et al. (2008), "Distributed Private Data Analysis: Simultaneously Solving How and What"

Our local algorithm for $(2+\eta)$-approximate core decomposition is obtained by applying the same thought process to the $(2+\eta)$-approximate algorithm of Dhulipala et al. [20]. The difference here is that, since the number of deletions is $O(\log^2 n)$, the counting mechanisms have to disclose far fewer prefix sums and, hence, achieve better accuracy.

Finally, Dhulipala et al. [150] showed that their exact core decomposition can be post-processed to obtain a 2-approximate densest subgraph with roughly the same additive error (in the centralized model). Similarly, Dhulipala et al. [20] implicitly proved this for their $(2+\eta)$-approximate core decomposition, obtaining a $(4+\eta)$-approximate densest subgraph (in the local model). We generalize their arguments to any approximation ratio, filling in details of the proof of Dhulipala et al. [150] (see the paragraph before Lemma 9.7.3). Specifically, we give a sufficient condition that allows a γ-approximate core decomposition to yield a 2γ-approximate densest subgraph with the same (asymptotic) additive error (in either model). Then we show that the condition is satisfied by our 1-approximate and $(2+\eta)$-approximate algorithms, hence obtaining 2-approximate $(4+\eta)$-approximate densest subgraph algorithms, respectively, with the same trade-offs.

9.4. Core Decomposition via Continual Counting

In this section, we describe an ε-edge differentially private mechanism for core decomposition on n-vertex graphs in the local model. The mechanism runs for at most n rounds and has $O(\varepsilon^{-1}\log^2 n)$ additive error with high probability. In the following, we refer to the coreness of a vertex that is returned by the mechanism as the *estimated* coreness of that vertex.

Theorem 9.4.1 *Given $\varepsilon > 0$, there is a local ε-edge differentially private mechanism $\mathscr{M}$ that returns an* estimate *of the coreness of each vertex in an n-vertex graph $G = (V, E)$ such that:*

- *$\mathscr{M}$ runs for at most n rounds in any execution, and*
- *for each vertex $v \in V$ with actual coreness $k(v)$, the estimated coreness $\tilde{k}(v)$ of v satisfies, with high probability:*

$$k(v) - \alpha \le \tilde{k}(v) \le k(v) + \alpha$$

where $\alpha = O(\varepsilon^{-1}\log n \log \Delta)$ and Δ is the maximum degree of G.

[158]: Matula et al. (1983), "Smallest-Last Ordering and clustering and Graph Coloring Algorithms"

Matula-Beck Peeling Algorithm:

1. compute the minimum degree, d, among all alive vertices in the graph,
2. iteratively kill vertices with degree at most d until none are left, and
3. report the coreness of each vertex killed in the preceding step as d.

At a high level, we implement a private version of the classic *peeling algorithm* of Matula and Beck [158] summarized on the next page. This algorithm iteratively deletes vertices in increasing order of their coreness by repeatedly performing the following steps, until the graph is empty.

To do so, a natural approach is for each vertex to send its *noisy* degree to the server in each round. That is, it sends its degree, plus some random noise to ensure privacy. Given these noisy degrees, the server can determine the set of vertices to be deleted in that round and broadcast it to every vertex. Upon receiving the server's message, each (surviving) vertex can determine which of its neighbors have been deleted and update its degree accordingly.

The main difficulty with this approach is generating the noisy degrees. In particular, a vertex may participate in many rounds (e.g., $\Omega(n)$ on a line graph) and, hence, send many noisy degrees. Naïvely using fresh noise each time requires too much noise to account for the privacy loss.

Our observation is that substantially less noise is needed if each vertex v uses a private *continual counting* mechanism, $\mathscr{C}_v$, instead. Specifically, v first generates a noisy degree by adding Laplace noise to its initial degree and sends this in the first round. If v is not deleted in a round, then it inserts the (absolute) *change* in its degree, as a result of zero or more of its neighbors being deleted in the round, into $\mathscr{C}_v$. Its initial noisy degree, minus the count that it receives from $\mathscr{C}_v$, is the noisy degree that it sends in the next round.

The pseudocode for the server and vertex algorithms are given below.

Algorithm 9.1: Server algorithm for coreness estimation on a graph $G = (V, E)$, in the local model.

$d \leftarrow 0$
foreach *round* $t = 1, 2, \ldots, n$ **do**
  receive noisy degree $\tilde{d}_t(v)$ from each vertex $v \in A_t := V \setminus (S_1 \cup \cdots \cup S_{t-1})$
  $d \leftarrow \max\{d, \min\{\tilde{d}_t(v) \mid v \in A_t\}\}$
  $S_t \leftarrow \{v \in A_t \mid \tilde{d}_t(v) \le d\}$
  broadcast S_t to all vertices in A_t and set estimate $\tilde{k}(v) = d$ for each vertex $v \in S_t$
  if $V \setminus (S_1 \cup \cdots \cup S_t) = \emptyset$ **then return** estimate vector $(\tilde{k}(v) : v \in V)$
end

Algorithm 9.2: User algorithm for coreness estimation on a graph $G = (V, E)$ in the local model; code for each vertex $v \in V$. We assume that v is given its set of neighbors, N_v, in G, as input and that v maintains an $\varepsilon/2$-differentially private continual counting mechanism, $\mathscr{C}_v$, supporting up to n adaptive insertions.

$\tilde{d}_1(v) \leftarrow |N_v| + \mathsf{Lap}(4/\varepsilon)$
foreach *round* $t = 1, \ldots, n$ **do**
  send $\tilde{d}_t(v)$ to server
  receive message S_t from server
  if $v \in S_t$ **then terminate**
  $\tilde{d}_{t+1}(v) \leftarrow \tilde{d}_1(v) - \mathscr{C}_v.\textsc{Insert}(|N_v \cap S_t|)$
end

9.4.1. Privacy

The key point is as follows: if the transcript (containing the sequence of outputs of each continual counting mechanism) is the same for the first $t-1$ rounds in two executions of the mechanism on neighboring graphs, then

the sequence of inputs inserted into each continual counting mechanism is nearly the same in both executions.

This kind of adaptive continual counting argument conditioning inductively on the past was also done in Chapter 7 for the first mechanism.

Observation 9.4.1 *Suppose the mechanism is executed on two graphs that differ by an edge e. If the transcript is the same in the first $t-1$ rounds in both executions, then the following holds.*

1. *The set of vertices deleted (i.e., the message broadcast by the server) in each round $r < t$ is the same in both executions.*
2. *For each vertex that is not an endpoint of e, its initial degree and subsequent changes in degree are the same in each round $r < t$ in both executions.*
3. *For each endpoint v of e, its initial degree differs by one in the two executions. It has a change in degree that differs (by one) in a round $r < t$ only if the other endpoint was deleted in round $r-1$, and v itself was not deleted in round $r-1$ (or earlier).*

Since the initial noisy degree is differentially private, the transcript of the first round is differentially private. By the preceding observation, to ensure that a transcript of subsequent rounds has roughly equal probability of occurring when the mechanism is executed on two neighboring graphs, it suffices for the counting mechanisms to have roughly equal probability of producing the same outputs, when executed on streams that *collectively* differ in at most one input (namely, at most one change in degree of an endpoint of the differing edge). A subtle point is that each counting mechanism needs to support *adaptive* insertions, as each input to the mechanism depends on its previous outputs.

We give a proof that is slightly different from the one in Chapter 7, but it is of a similar flavor.

Lemma 9.4.2 *If the continual counting mechanism of each vertex is $\varepsilon/2$-differentially private and supports up to n adaptive insertions, then the transcript is ε-edge differentially private.*

Proof. We view a transcript as a vector $\pi = (\pi_1, \ldots, \pi_n)$ of functions $\pi_t : V \to \mathbb{R} \cup \{\perp\}$, for $t \in [n]$, where $\pi_t(v)$ is the message sent by vertex $v \in V$ in round t, or $\perp$ if it sends no message. In particular, since the message, S_t, broadcast by the server in round t is a deterministic function of $(\pi_1, \ldots, \pi_t)$, we omit it from the transcript.

Consider two graphs G and G' that differ in an edge e. Let Π and Π' denote the (random) transcript of the mechanism when executed on G and G', respectively. We claim that for any set of (valid) transcripts $\mathscr{A}$,

$$\Pr(\Pi \in \mathscr{A}) \leq e^{\varepsilon} \cdot \Pr(\Pi' \in \mathscr{A}).$$

Indeed, fix a transcript $\pi \in \mathscr{A}$. We will show that $\Pr(\Pi = \pi) \leq e^{\varepsilon} \Pr(\Pi' = \pi)$. We split the analysis into two steps, each consisting of one mechanism that we analyze separately:

Arguing per transcript suffices in this case since we are working with a finite set for pure differential privacy.

1. The mechanism M_1 that discloses $\tilde{d}_1(v)$ for each vertex v, and
2. the mechanism M_2 that discloses $\tilde{d}_t(v)$ for each vertex v and all $t > 1$.

Mechanism M_1. Consider a vertex $v \in V$. If $v \notin e$, then the degree of v is the same in both graphs, while for the two endpoints of e, the difference is exactly 1. Consider the vector containing the degrees of all vertices, in some fixed order. Then the ℓ_1-norm of the difference of the vector for G and the vector for G' is 2. Thus, adding Laplace noise with parameter $\varepsilon/4$ to each entry in the vector guarantees that M_1 is $\varepsilon/2$-edge differentially private.

This is a simple Laplace mechanism.

Mechanism M_2. For each vertex $v \in V$, let

$$p_v := \prod_{t=2}^{n} \Pr(\Pi_t(v) = \pi_t(v) \mid \wedge_{r=1}^{t-1} \Pi_r = \pi_r) \text{ and}$$
$$p'_v := \prod_{t=2}^{n} \Pr(\Pi'_t(v) = \pi_t(v) \mid \wedge_{r=1}^{t-1} \Pi'_r = \pi_r).$$

By definition, $\Pi_1(v)$ is the value returned by M_1. For $t > 1$, conditioned on the transcript of the first $t-1$ rounds being $(\pi_1, \dots, \pi_{t-1})$ in both executions, the sequence of messages, $S_1, \dots, S_{t-1}$, broadcast by the server in the first $t-1$ rounds is the same in both executions. If v participates in round t, i.e., $v \notin S_1 \cup \cdots \cup S_{t-1}$, then $\Pi_t(v)$ is the count returned by $\mathscr{C}_v$ immediately after v inserts the input $c_t(v) := -|N_v \cap S_{t-1}|$. Otherwise, $\Pi_t(v) = \perp$.

$c_t(v)$ is the $(t-1)$-th change in degree of v in the execution on G

It follows that p_v is the probability that $\mathscr{C}_v$ returns the sequence of outputs $(\pi_1(v), \pi_2(v), \dots)$ on the (adaptive) sequence of inputs $(c_1(v), c_2(v), \dots)$. Defining $c'_1(v), c'_2(v), \dots$ similarly for the execution on G', we have that p'_v is the probability that $\mathscr{C}_v$ returns the sequence of outputs $(\pi_1(v), \pi_2(v), \dots)$ on the sequence of inputs $(c'_1(v), c'_2(v), \dots)$.

By Observation 9.4.1, the inputs to $\mathscr{C}_v$ are the same in both executions conditioned on the previous outputs being the same, i.e., for each vertex $v \notin e$,

$$(c_1(v), c_2(v), \dots) = (c'_1(v), c'_2(v), \dots).$$

By coupling the random bits of each vertex $v \notin e$ in the two executions, this implies that each output of $\mathscr{C}_v$ has the same probability of occurrence under both executions (conditioned on the previous outputs being equal). It follows that $p_v = p'_v$ for $v \notin e$ and, hence,

$$\prod_{v \notin e} p_v = \prod_{v \notin e} p'_v.$$

On the other hand, for each endpoint $v \in e$, and $t > 1$, $c_t(v) \neq c'_t(v)$ only if S_{t-1} contains the other endpoint $u \in e$, but not v. In this case, $(c_2(v), c_3(v), \dots)$ and $(c'_2(v), c'_3(v), \dots)$ differ by one element, namely

$$|c_t(v) - c'_t(v)| = 1 \quad \text{and} \quad (c_2(u), c_3(u), \dots) = (c'_2(u), c'_3(u), \dots).$$

If this case does not happen, i.e., if u and v belong to the same set S_{t-1}, then $(c_2(v), c_3(v), \dots)$ and $(c'_2(v), c'_3(v), \dots)$ do not differ. Since $\mathscr{C}_u$ and $\mathscr{C}_v$ are $\varepsilon/2$-differentially private, it follows that $p_u = p'_u$ and $p_v \leq e^{\varepsilon/2} \cdot p'_v$. Hence,

$$\prod_{v \in e} p_v \leq e^{\varepsilon/2} \prod_{v \in e} p'_v.$$

Since each vertex independently generates its message in each round t,

$$\begin{aligned}
\Pr(\Pi = \pi) &= \prod_{v \in V} \prod_{t=1}^{n} \Pr\left(\Pi_t(v) = \pi_t(v) \mid \wedge_{r=1}^{t-1} \Pi_r = \pi_r\right) \\
&= \prod_{v \notin e} p_v \times \prod_{v \in e} p_v \\
&\leq \prod_{v \notin e} p'_v \times e^{\varepsilon/2} \cdot \prod_{v \in e} p'_v \\
&= e^{\varepsilon/2} \cdot \prod_{v \in V} \prod_{t=1}^{n} \Pr\left(\Pi'_t(v) = \pi_t(v) \mid \wedge_{r=1}^{t-1} \Pi'_r = \pi_r\right) \\
&= e^{\varepsilon/2} \cdot \Pr(\Pi' = \pi).
\end{aligned}$$

This shows that M_2 is $\varepsilon/2$-edge differentially private. By simple composition, the complete mechanism, which consists of the output of M_1 and M_2, is ε-edge differentially private. □

9.4.2. Accuracy

The noisy degrees sent to the server may differ significantly from the actual degrees. Hence, in each round, the server may incorrectly delete some vertices or fail to delete others. We show that if there is a bound on the error of the noisy degrees (of all vertices and in all rounds), then the same bound holds on the error of the coreness estimates.

This shows that estimating the core numbers is smooth with respect to noise in the degree estimates.

Lemma 9.4.3 *Given a graph $G = (V, E)$, if each noisy degree differs from its corresponding actual degree by at most α, then, for every vertex $v \in V$ with actual coreness $k(v)$, the estimated coreness $\tilde{k}(v)$ of v satisfies*

$$k(v) - \alpha \leq \tilde{k}(v) \leq k(v) + \alpha.$$

Furthermore, every vertex in $G[U]$ has (induced) degree at least $\tilde{k}(v) - \alpha$, where U is the set of all vertices u with estimated coreness $\tilde{k}(u) \geq \tilde{k}(v)$.

This second statement of the lemma is only needed for the accuracy proof of the approximate densest subgraph algorithm in Section 9.7.2.

Proof. Fix any vertex $v \in V$. Let t be the round in which v is deleted, let d_t (resp. $\tilde{d}_t(v)$) be the value of the variable d (resp. message from vertex v) stored by (resp. received by) the server at the end of round t, and let r be the first round in which the server sets d to d_t. By definition, $\tilde{k}(v) = d_t$, $r \leq t$, and d_t is the minimum noisy degree sent in round r, i.e., $\tilde{d}_r(u) \geq d_t$ for each $u \in U$, where U is the set of non-deleted vertices at the start of round r. Since r is the first round where d is set to d_t and d is non-decreasing, it follows that U is the set of all vertices u with estimated coreness $\tilde{k}(u) \geq d_t$. By definition, $v \in U$ and every vertex in $G[U]$ has (*actual*) induced degree at least $d_t - \alpha$ since $\tilde{d}_r(u) \geq d_t$ and the noisy degree differs in every round at most α from its corresponding degree. By definition, G contains a subgraph H containing v in which every vertex has induced degree at least $k(v)$. Note that $k(v)$ is the largest integer for which this holds, so $k(v) \geq d_t - \alpha$ and, hence, $\tilde{k}(v) = d_t \leq k(v) + \alpha$. This shows the second inequality of the lemma.

Consider the first vertex, u, of H to be deleted. Let t' be the round in which u is deleted and let $d_{t'}$ be the value of d stored at the end of round t'. By definition, no vertex in H (in particular, v) has been deleted at the start of round t'. Hence, $t' \leq t$ and u sends a noisy degree $\tilde{d}_{t'}(u)$ that is at least $k(v) - \alpha$. On the other hand, since u is deleted in round t', the noisy degree that it sends in that round is at most $d_{t'}$. Since d is non-decreasing, it follows that $k(v) - \alpha \leq d_{t'} \leq d_t = \tilde{k}(v)$. This shows the first inequality. □

By Laplace tail bounds with $\beta = 1/n^{\Omega(1)}$ and a union bound over all vertices, with high probability,

$$\deg_G(v) - O(\varepsilon^{-1} \log n) \leq \tilde{d}_1(v) \leq \deg_G(v) + O(\varepsilon^{-1} \log n)$$

for all $v \in V$. Hence, the additive error of $\tilde{d}_{t+1}(v)$ is $O(\varepsilon^{-1} \log n)$, plus the error of the continual counting mechanism, with high probability. Note that the length of any input sequence is bounded by $T = n$ and its prefix sums are bounded by $n_T = \Delta$. Hence, taking $\beta = n^{-\Omega(1)}$, the sparse-vector counting mechanism has additive error $O(\varepsilon^{-1} \log n \log \Delta)$ with high probability (Theorem 3.2.3). Combining this with Lemma 9.4.3 immediately yields:

Lemma 9.4.4 *With high probability, for every vertex $v \in V$, the estimated coreness of v differs from the actual coreness of v by at most $O(\varepsilon^{-1} \log n \log \Delta)$ when using the sparse-vector counting mechanism.*

9.4.3. Further Applications

Dhulipala et al. [20] gave a local ε-edge differentially private mechanism by building on Liu et al. [159] for approximate core decomposition. We show in Section 9.7.1 that our framework can be used in this setting to obtain:

[159]: Liu et al. (2022), "Parallel Batch-Dynamic Algorithms for k-Core Decomposition and Related Graph Problems"

Theorem 9.4.5 *Given $\varepsilon, \eta > 0$, there is a local ε-edge differentially private mechanism $\mathscr{M}$ that returns an estimate of the coreness of each vertex in an n-vertex graph $G = (V, E)$ such that:*

- *$\mathscr{M}$ runs for $O(\log_{1+\eta} n \log n)$ rounds in any execution, and*
- *for each vertex $v \in V$ with actual coreness $k(v)$, the estimated coreness $\tilde{k}(v)$ of v satisfies:*

$$k(v) - \alpha \leq \tilde{k}(v) \leq (2+\eta)k(v) + \alpha\,,$$

where $\alpha = O(\varepsilon^{-1} \log n \log \log_{1+\eta} n)$ with probability $1 - n^{-\Omega(1)}$.

We also show in Section 9.7.2 how to obtain the following guarantees for densest subgraph using our results for coreness approximation.

This will essentially follow from the fact that the maximum core is a 2-approximation to the densest subgraph, with a little bit of technical details to fill in.

Theorem 9.4.6 *Given $\varepsilon, \eta > 0$, there are local ε-edge differentially private mechanisms $\mathscr{M}$ and $\mathscr{M}'$ that return subsets of vertices U and U' from a given n-vertex graph G, respectively, such that:*

- *$\mathscr{M}$ and $\mathscr{M}'$ run for at most n and $O(\log_{1+\eta} n \log \log n)$ rounds, respectively, in any execution.*
- *With probability $1 - n^{-\Omega(1)}$, the density of $G[U]$ and $G[U']$ is at least $\rho^*/2 - \alpha$ and $\rho^*/(4+\eta) - \alpha'$, respectively, where ρ^* is the maximum density of any subgraph of G, $\alpha = O(\varepsilon^{-1} \log n \log \Delta)$, $\alpha' = O(\varepsilon^{-1} \log_{1+\eta} n \log \log n)$, and Δ is the maximum degree of G.*

9.5. Lower bounds for core decomposition

Since the local model is stronger, the centralized lower bound carries over to the local model.

We first give a lower bound in the centralized model and then give a lower bound for the local model. Here, $\varepsilon > 0$ is a positive constant.

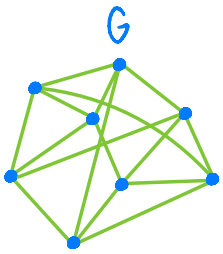
Theorem 9.5.1 *Given a constant $\gamma \geq 1$, and letting V be a set of n vertices, suppose that $\mathscr{M}$ is an ε-edge differentially private mechanism in the centralized model that estimates the coreness of every vertex in a given graph G on V such that, for all vertices $v \in V$ with actual coreness $k(v)$, the estimated coreness $\tilde{k}(v)$ of v satisfies*

$$\gamma^{-1} k(v) - \alpha \leq \tilde{k}(v) \leq \gamma k(v) + \alpha$$

for all $v \in V$ simultaneously with probability at least p. Then

$$\alpha = \Omega(\gamma^{-1} \log(np)).$$

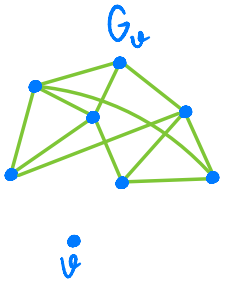
Proof. Let $d = \lceil (2\alpha+1)\gamma \rceil$ and let G be any $(d+1)$-regular graph on V. For each vertex $v \in V$, let G_v be the same as G, except that all $d+1$ edges incident to v are removed.



Observe that, in G_v, v has coreness 0, while all other vertices $u \neq v$ have coreness at least d. Let $\mathscr{C}_v$ be the set of all coreness estimate vectors $\tilde{k}$ such that $\tilde{k}(v) \leq \alpha$ and $\tilde{k}(u) > \alpha$ for all $u \neq v$. Observe that $\mathscr{C}_v$ and $\mathscr{C}_u$ are disjoint for $u \neq v$. Furthermore, since $\gamma^{-1}d - \alpha = \alpha + 1$, $\mathscr{M}(G_v)$ must return $\tilde{k}(v) \leq \alpha$ and for all $u \neq v$, $\tilde{k}(u) \geq \alpha + 1$, i.e., an estimate in $\mathscr{C}_v$ with probability $\geq p$.

Since G and G_v differ by $d+1$ edges and $\mathscr{M}$ satisfies ε-edge differential privacy, we have that

$$\begin{aligned}\Pr\left(\mathscr{M}(G) \in \mathscr{C}_v\right) &\geq e^{-\varepsilon(d+1)} \cdot \Pr\left(\mathscr{M}(G_v) \in \mathscr{C}_v\right) \\ &\geq e^{-\varepsilon(d+1)} \cdot p.\end{aligned}$$



Since $\mathscr{C}_u$ and $\mathscr{C}_v$ are disjoint for $u \neq v$,

$$\begin{aligned}1 &\geq \Pr\left(\bigcup_{v \in V} \left(\mathscr{M}(G) \in \mathscr{C}_v\right)\right) \\ &= \sum_{v \in V} \Pr\left(\mathscr{M}(G) \in \mathscr{C}_v\right) \\ &\geq n \cdot e^{-\varepsilon(d+1)} \cdot p.\end{aligned}$$

By rearranging,

$$\lceil (2\alpha+1)\gamma \rceil = d \geq \ln(np) - 1.$$

Therefore, $\alpha = \Omega(\gamma^{-1} \log(np))$. □

Let V be a set of $2n+1$ vertices and let $x \in V$ be a fixed vertex. Consider any ε-edge differentially private mechanism $\mathscr{M}$ that *non-interactively* (i.e., in a single round) estimates the coreness of x in a given graph on V, in the local model. We show that there is a large family of graphs on which $\mathscr{M}$ has constant probability of returning an estimate with $\Omega(\sqrt{n})$ error for x.

Theorem 9.5.2 *Given a constant $\varepsilon > 0$, there exists a constant $0 < \eta < \frac{1}{2}$ such that the following holds: Suppose that $\mathscr{M}$ is a non-interactive ε-edge differentially private local mechanism that estimates the coreness of a fixed vertex, x, in an arbitrary n-vertex graph such that, if $k(x)$ is the actual coreness of x, then the estimated coreness $\tilde{k}(x)$ of x satisfies:*

$$k(x) - \alpha \leq \tilde{k}(x) \leq k(x) + \alpha \text{ with probability at least } \frac{1}{2} + \eta.$$

Then there is a family of n-vertex graphs of size $2^{\Omega(n)}$ on which $\alpha = \Omega(\sqrt{n})$.

Our approach is to reduce to a known lower bound on the error of privately answering a linear number of random *inner product queries* on a secret dataset $X \in \{0,1\}^n$. Here, two datasets X and X' are *neighboring* if they differ in at most one coordinate, a query is specified by a vector $Q \in \mathbb{R}^n$, and the *error* of a response r to query Q is $|r - \langle Q, X \rangle|$. Eden et al. [154] were the first to use this approach to prove lower bounds in the local model.

[154]: Eden et al. (2023), "Triangle Counting with Local Edge Differential Privacy"

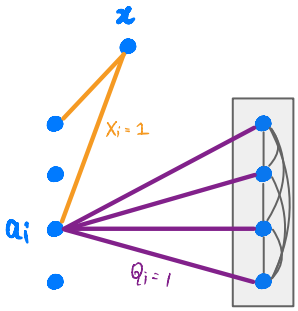
Roughly speaking, the lower bound says that no private mechanism (with a trusted curator) can answer $O(n)$ random inner product queries in $\{-1,1\}^n$ so that, with constant probability, a large fraction of the answers have $o(\sqrt{n})$ error. (Otherwise, an "attacker" can use such queries to "reconstruct" X with high accuracy, violating privacy.) In Section 9.8, we formally state the lower bound and use it to prove the following modified variant, adapted to inner

product queries in $\{0,1\}^n$. The idea is to convert a mechanism answering inner product queries in $\{0,1\}^n$ into a mechanism answering inner product queries in $\{-1,1\}^n$ with roughly the same error and privacy loss.

Theorem 9.5.3 *Given $\varepsilon > 0$ and $\delta \in [0, 0.05]$, there is a constant $\eta \in (0, 0.5)$ such that no (ε, δ)-differentially private mechanism can answer $m = O(n)$ random inner product queries in $\{0,1\}^n$ on a secret dataset $X \in \{0,1\}^n$ such that, with probability at least $\Omega(\sqrt{\delta})$, a $(0.5+\eta)$-fraction of its answers have $o(\sqrt{n})$ error.*

To apply Theorem 9.5.3, we construct a 2ε-differentially private mechanism $\mathcal{N}$ that answers m random inner product queries in $\{0,1\}^n$. For each query, $\mathcal{N}$ simulates $\mathcal{M}$ on a *query graph* in which the coreness of x is roughly the intended answer to the query. We show that, with constant probability, a large fraction of $\mathcal{N}$'s responses will have the same error as $\mathcal{M}$. Therefore, $\mathcal{M}$ has $\Omega(\sqrt{n})$ error with constant probability.

Query Graphs. Fix a partition (A, B) of $V \setminus \{x\}$ with $|A| = |B|$ and an enumeration, $a_1, \ldots, a_n$, of the vertices in A. Let $Q \in \{0,1\}^n$ be an arbitrary inner product query. The *query graph* for Q on dataset X is the graph $G_X(Q)$ on V defined as follows.



1. For all $i \in [n]$, $X_i \in \{0,1\}$ indicates whether a_i is adjacent to x and $Q_i \in \{0,1\}$ indicates whether a_i is adjacent to either *no* vertex in B or to *every* vertex in B.
2. The vertices of B form a clique, which x is not adjacent to.

The idea is that Q is used to bound the coreness of vertices in A. Specifically, if $Q_i = 1$, then a_i is adjacent to every vertex in B and, hence, has coreness at least $n-1$. Otherwise, a_i can only be adjacent to x, so it has coreness at most 1. Thus, the coreness of x is roughly the number of neighbors $a_i \in A$ such that $Q_i = 1$, i.e., $\langle Q, X \rangle$. (If Q is the all-zero vector, then the coreness of x may still be 1.)

Lemma 9.5.4 *The coreness of x in $G_X(Q)$ is either $\langle Q, X \rangle$ or $\langle Q, X \rangle + 1$.*

Proof. Let $A' = \{a_i \in A \mid Q_i = X_i = 1\}$. Notice that $|A'| = \langle Q, X \rangle$. If $|A'| = 0$, then every neighbor (if any) of x has degree 1. Hence, the coreness of x is at most $1 = \langle Q, X \rangle + 1$.

Now suppose $|A'| \neq 0$. We will show that the coreness of x equals $|A'| = \langle Q, X \rangle$. By construction, every vertex in A' is adjacent to x and every vertex in B. It follows that $A' \cup B \cup \{x\}$ induces a subgraph of $G_X(Q)$ in which every vertex has (induced) degree $\geq |A'| \geq 1$. Thus, the coreness of x is $\geq |A'|$.

Next consider any set of vertices S containing x. If S contains a vertex $a_i \in A \setminus A'$ that is adjacent to x, then $Q_i = 0$ and the degree of a_i in the subgraph of $G_X(Q)$ induced by S is $1 \leq |A'|$. Otherwise, every neighbor of x in the subgraph of $G_X(Q)$ induced by S is in A' and, hence, the induced degree of x is $\leq |A'|$. Hence, in either case the coreness of x is $\leq |A'|$. □

Answering Random Queries. Naïvely answering queries by simulating $\mathcal{M}$ on each of the corresponding query graphs leads to prohibitively large privacy loss. Our key observation is that the vertices whose neighborhoods in the query graphs depend on the secret dataset X, namely $\{x\} \cup A$, only have a few possible neighborhoods among all query graphs on X. This suggests

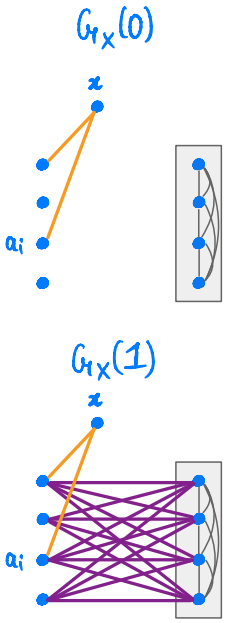
that, to reduce the privacy loss, we can generate only a few messages for these vertices and *reuse* them in all simulations.

Observation 9.5.1 *The following holds for any secret dataset $X \in \{0,1\}^n$.*

1. *The neighbors of x are the same in all query graphs on X.*
2. *Each vertex in A only has two possible neighborhoods among all query graphs on X (namely, $\{x\}$ and $B \cup \{x\}$, if $X_i = 1$, and $\emptyset$ and B otherwise).*
3. *For all datasets X, the neighborhood of each vertex $b \in B$ in the query graph $G_X(Q)$ is a function of Q, i.e., $(B \setminus \{b\}) \cup \{a_i \in A \mid Q_i = 1\}$.*

Specifically, our mechanism $\mathcal{N}$ answers m random inner product queries $Q^{(1)}, \ldots, Q^{(m)} \in \{0,1\}^n$ on a secret dataset $X \in \{0,1\}^n$ as follows.

1. Let $\mathbf{0}$ and $\mathbf{1} \in \{0,1\}^n$ be the all-zeros and all-ones vectors, respectively.
   a) Run the user algorithm of x (specified by $\mathscr{M}$) on its neighborhood in $G_X(\mathbf{0})$ to obtain the message π_x.
   b) For each $i \in [n]$: run the user algorithm of a_i (specified by $\mathscr{M}$) on its neighborhood in $G_X(\mathbf{0})$ and $G_X(\mathbf{1})$ to obtain the messages $\pi_{a_i}(0)$ and $\pi_{a_i}(1)$, respectively.
2. For each query $Q^{(j)}$, *simulate* a run of $\mathscr{M}$ on $G_X(Q^{(j)})$ as follows:
   a) Run the user algorithm of each vertex $b \in B$ (specified by $\mathscr{M}$) on its neighborhood in the query graph $G_X(Q^{(j)})$ to obtain the message $\pi_b^{(j)}$.
   b) Run the server's algorithm (specified by $\mathscr{M}$) on the transcript
   $$\{\pi_x\} \cup \{\pi_{a_i}(Q_i^{(j)}) \mid i \in [n]\} \cup \{\pi_b^{(j)} \mid b \in B\}$$
   to obtain an estimate $\tilde{k}_j(x)$ of the coreness of vertex x in the query graph $G_X(Q^{(j)})$.
3. Return $(\tilde{k}_j(x) : j \in [m])$.



Analysis. Since we run the user algorithm of each vertex in $\{x\} \cup A$ on its respective neighborhood in at most two graphs and $\mathscr{M}$ is ε-edge differentially private, the messages generated in the first step are collectively 2ε-edge differentially private. Since the pairs of graphs $G_X(\mathbf{0}), G_{X'}(\mathbf{0})$ and $G_X(\mathbf{1}), G_{X'}(\mathbf{1})$ each differ in at most one edge when datasets X and X' are neighboring, it follows that the first step is 2ε-differentially private. The other steps can be viewed as simply post-processing the outputs (messages) of the first step. Therefore, the entire procedure is 2ε-differentially private.

Lemma 9.5.5 *$\mathcal{N}$ is 2ε-differentially private.*

Proof. Let $X, X' \in \{0,1\}^n$ be two datasets that differ only at the i-th coordinate. Consider any valid combination of messages,

$$\{\pi_x\} \cup \{\pi_{a_j}(0) \mid j \in [n]\} \cup \{\pi_{a_j}(1) \mid j \in [n]\}$$

generated in the first step. Since X and X' differ only at the i-th coordinate, the neighborhoods of x in $G_X(\mathbf{0}), G_{X'}(\mathbf{0})$ differ in only one edge, namely $\{x, a_i\}$. Similarly, for the neighborhoods of a_i in the pairs of graphs $G_X(\mathbf{0}), G_{X'}(\mathbf{0})$ and $G_X(\mathbf{1}), G_{X'}(\mathbf{1})$. The neighborhoods of the other vertices

in A are the same in each pair of graphs $G_X(\mathbf{0}), G_{X'}(\mathbf{0})$ and $G_X(\mathbf{1}), G_{X'}(\mathbf{1})$. It follows that the ratio of the probabilities of seeing the (partial) transcript

$$\{\pi_x\} \cup \{\pi_{a_j}(0) \mid j \in [n]\}$$

on the graphs $G_X(\mathbf{0}), G_{X'}(\mathbf{0})$ is at most e^ε, since the transcript is ε-edge differentially private. Similarly, for the ratio of the probabilities of seeing the (partial) transcript

$$\{\pi_{a_j}(1) \mid j \in [n]\}$$

on the graphs $G_X(\mathbf{1}), G_{X'}(\mathbf{1})$. Thus, the ratio of the probabilities of seeing the entire combination of messages in the first step on datasets X and X' is at most $e^\varepsilon \cdot e^\varepsilon = e^{2\varepsilon}$ and the first step is 2ε-differentially private. The remaining steps do not require knowledge of X and, hence, only serve as post-processing. Therefore, $\mathcal{N}$ is 2ε-differentially private. □

Lemma 9.5.4 implies that $\mathcal{N}$ produces a response to a query with additive error $> \alpha + 1$ only if the simulation of $\mathcal{M}$ on the corresponding query graph produces an estimate of the coreness of x that has additive error exceeding α. Since the messages of vertices in $\{x\} \cup A$ are reused, the simulations are *not* independent. Nonetheless, Markov's inequality can be used to bound the probability that a large number of simulations have error exceeding α.

Lemma 9.5.6 *Suppose that $\mathcal{M}$ has error exceeding α with probability at most β. Then, with probability at least $1 - 1/\gamma$, at least a $(1 - \gamma\beta)$-fraction of the responses of $\mathcal{N}$ have error at most $\alpha + 1$.*

Proof. For $j \in \{1, \ldots, m\}$, let $X_j \in \{0, 1\}$ be the indicator random variable for whether the response to the j-th query has error exceeding $\alpha + 1$, i.e.,

$$|\tilde{k}_j(x) - \langle Q^{(j)}, X\rangle| > \alpha + 1.$$

Let $k_j(x)$ be the actual coreness of x in $G_X(Q^{(j)})$. Then

$$\begin{aligned} |\tilde{k}_j(x) - \langle Q^{(j)}, X\rangle| &\leq |\tilde{k}_j(x) - k_j(x)| + |k_j(x) - \langle Q^{(j)}, X\rangle| && \text{(by triangle inequality)} \\ &\leq |\tilde{k}_j(x) - k_j(x)| + 1. && \text{(by Lemma 9.5.4)} \end{aligned}$$

Since $\mathcal{M}$ has error exceeding α with probability at most β, it follows that $E[X_j] \leq \beta$. Hence, we have

$$E[\sum_{j=1}^{m} X_j] = \sum_{j=1}^{m} E[X_j] \leq \beta m, \quad \text{and} \qquad \text{(by linearity of expectation)}$$

$$\Pr(\sum_{i=1}^{m} X_i \geq \gamma\beta m) \leq 1/\gamma. \qquad \text{(by Markov's inequality)}$$

Therefore, with probability $1 - 1/\gamma$, at least $(1 - \gamma\beta)m$ responses of $\mathcal{N}$ have error at most $\alpha + 1$. □

To summarize, we have shown that, if $\mathcal{M}$ estimates the coreness of a fixed vertex with error exceeding α with probability at most $\beta = \frac{1}{2} - \eta$, then with constant probability (say $1 - 1/1.001 \approx 0.001$), a large fraction ($1 - 1.001\beta \approx \frac{1}{2} + \eta$) of the responses of $\mathcal{M}$ have error at most $\alpha + 1$. Since $\mathcal{N}$ is 2ε-differentially private, it is $(2\varepsilon, \delta)$-differentially private for any $\delta > 0$. Therefore, by the contrapositive of Theorem 9.5.3, $\alpha = \Omega(\sqrt{n})$.

9.6. Formal Definition of Differential Privacy in the Local Model

We formally define our model in this section and show that private local mechanisms can be emulated using local randomizers, which shows that our results apply to the model used in prior work [20, 153, 154].

A *local mechanism* $\mathscr{M}$ is defined by a pair of functions $f : \mathscr{T} \times \mathscr{L} \times \{0,1\}^* \to \Sigma^* \times \mathscr{L}$ and $g : \mathscr{T} \times \{0,1\}^* \to \Sigma^*$, where $\mathscr{T}$ is the set of all possible $\mathscr{M}$-transcripts[5], Σ^* is the set of all possible messages, $\mathscr{L}$ is the set of all possible local memory states, and $\{0,1\}^*$ is the set of all binary strings. An *execution* of a local mechanism $\mathscr{M} = \{f, g\}$ on an input graph $G = (V, E)$ generates a *$\mathscr{M}$-transcript* $\hat{\tau}$ as follows. Initially, the local memory of each user v is $\ell_v^0 := (v, N_v)$, where N_v is the set of neighbors of v in G, and the (*0-round*) $\mathscr{M}$-transcript $\hat{\tau}$ is empty. At the start of the $(t+1)$-th *round*, each user v simultaneously generates fresh random bits $\gamma_v^{t+1} \in \{0,1\}^*$ and evaluates $(\sigma_v^{t+1}, \ell_v^{t+1}) := f(\hat{\tau}, \ell_v^t, \gamma_v^{t+1})$. Then the local memory of each user v is set to ℓ_v^{t+1} and $\hat{\tau}$ is updated to $\hat{\tau} \odot (\sigma_v^{t+1} : v \in V)$, where $\odot$ is the string concatenation operation. Subsequently, the server generates fresh random bits $\gamma_s^{t+1} \in \{0,1\}^*$ and $\hat{\tau}$ is updated to $\hat{\tau} \odot g(\hat{\tau}, \gamma_s^{t+1})$. (At this point, we say $\hat{\tau}$ is a $(t+1)$ *$\mathscr{M}$-round* transcript.) If $g(\hat{\tau}, \gamma_s^{t+1})$ is not the empty string, then the next round begins. Otherwise, the execution ends and the resulting $\hat{\tau}$ is a *complete* $\mathscr{M}$-transcript (on input G).

5: To distinguish between transcripts arising from local mechanisms and protocols using local randomizers, we will use the terminology $\mathscr{M}$-transcript and $\mathscr{A}$-transcript, respectively, in this section.

We say that $\mathscr{M}$ is *ε-edge differentially private* if the output (transcript) of $\mathscr{M}$ is ε-differentially private on neighboring graphs. We say that $\mathscr{M}$ is *memoryless* if f is the identity function on its second argument, i.e., for all $\hat{\tau}, \ell, \gamma$, there exists σ such that $f(\hat{\tau}, \ell, \gamma) = (\sigma, \ell)$. In particular, in a memoryless local mechanism, the local memory of each user v is always set to (v, N_v).

Our definition implicitly assumes that the server includes each message it has received (i.e., the transcript so far) in its next "message" to the users (which is $\hat{\tau} \odot g(\hat{\tau}, \gamma_s^{t+1})$). This is why each user's "message" is a function of the transcript so far, even if each user is memoryless. Similarly, for each user's "message" to the server. Hence, since the server does not have any private data, it does not need local memory. In particular, its message can include the random string γ_s^{t+1} it sampled.

An ε-differentially private memoryless local mechanism $\mathscr{M} = \{f, g\}$ in our model can be modeled by a protocol $\mathscr{A}$, as defined by Dhulipala et al. [20] and presented earlier. The main difference is that there is no server in the latter model and $\mathscr{A}$ needs to map a partial $\mathscr{A}$-transcript to an assignment of local randomizers to the participating users of the next round, one per user. To handle this, we hard-code the server's message in $\mathscr{M}$ that is sent at the end of the previous round into the local randomizer assigned to each user in the current round of $\mathscr{A}$. To simulate the random bits of the server, one fixed user u sends additional random bits, which allows the users to agree on the random string generated by the server in the previous round.

More formally, let $\mathscr{M} = \{f, g\}$ be any ε-edge differentially private memoryless local mechanism. We define a protocol $\mathscr{A}$ in the local randomizer model so that each t-round $\mathscr{A}$-transcript *corresponds* to a t-round $\mathscr{M}$-transcript. Indeed, the (empty) 0-round $\mathscr{A}$-transcript *corresponds* to the (empty) 0-round $\mathscr{M}$-transcript. Now consider any t-round $\mathscr{A}$-transcript τ that corresponds to a t-round $\mathscr{M}$-transcript $\hat{\tau}$. If $\hat{\tau}$ is non-empty and the last message of the server in $\hat{\tau}$ is the empty string, then we define $\mathscr{A}(\tau) = \perp$. Otherwise, we define $\mathscr{A}(\tau) = (S_U^{t+1}, S_R^{t+1}, S_\varepsilon^{t+1})$, where

- the set S_U^{t+1} of participating users is V,

- the local randomizer $S_R^{t+1}(v)$ assigned to user v generates fresh random bits $\gamma_v \in \{0,1\}^*$ (and $\gamma_s \in \{0,1\}^*$, if $v = u$) and then outputs $S_O^{t+1}(v) = \sigma_v$ for $v \neq u$ and $S_O^{t+1}(u) = (\sigma_u, \gamma_s)$, where $(\sigma_v, (v, N_v)) = f(\hat{\tau}, (v, N_v), \gamma_v)$, i.e., σ_v is the first parameter of $f(\hat{\tau}, (v, N_v), \gamma_v)$[6], and
- the privacy parameter of $S_R^{t+1}(v)$ is $S_\varepsilon^{t+1}(v) = \varepsilon$.

6: Note the use of the $\mathscr{M}$-transcript $\hat{\tau}$.

In this case, the resulting $(t+1)$-round $\mathscr{A}$-transcript

$$\tau' = \tau \odot (S_U^{t+1}, S_R^{t+1}, S_\varepsilon^{t+1}, S_O^{t+1})$$

corresponds to the $(t+1)$-round $\mathscr{M}$-transcript

$$\hat{\tau}' = \hat{\tau} \odot (\sigma_v : v \in V) \odot g(\hat{\tau} \odot (\sigma_v : v \in V), \gamma_s),$$

The set $\mathscr{R}$ of local randomizers used by $\mathscr{A}$ essentially consists of all functions f with the first parameter hard-coded to be any value of $\mathscr{T}$, which is used to encode the corresponding $\mathscr{M}$-transcript and, in particular, the messages sent by the server to the users, as given by g. Furthermore, we emphasize that, in the specification of $\mathscr{A}(\tau')$, the corresponding $\mathscr{M}$-transcript $\hat{\tau}'$ is used to determine if $\mathscr{A}(\tau') = \perp$ and, if $\mathscr{A}(\tau') \neq \perp$, to determine the output of each local randomizer. Since (the output of) each local randomizer is chosen as a function of the corresponding $\mathscr{M}$-transcript, $\mathscr{A}$ is ε-edge dp.

9.7. Applications of Continual Counting

9.7.1. Approximate Core Decomposition

[159]: Liu et al. (2022), "Parallel Batch-Dynamic Algorithms for k-Core Decomposition and Related Graph Problems"

Let $\varepsilon, \eta > 0$ be any positive constants. Dhulipala et al. [20] gave a local ε-edge differentially private mechanism by building on Liu et al. [159] for approximate core decomposition on n-vertex graphs $G = (V, E)$ with the following one-sided *multiplicative* error guarantee: for every vertex $v \in V$ with actual coreness $k(v)$, the *estimated* coreness $\tilde{k}(v)$ of v (i.e., as reported by the mechanism) satisfies:

$$k(v) - \alpha \leq \tilde{k}(v) \leq (2+\eta)k(v) + \alpha\,,$$

For $\eta = 1$, this is $O(\log^3 n)$ error.

where $\alpha = O(\varepsilon^{-1} \log_{1+\eta} n \log^2 n)$ with probability $1 - n^{-\Omega(1)}$.

Their mechanism has $T = O(\log_{1+\eta} n \log n)$ rounds, divided into $O(\log n)$ disjoint *phases*, each consisting of $O(\log_{1+\eta} n)$ consecutive rounds. In each round, the non-deleted vertices send their current noisy degrees to the server. Each noisy degree is generated by adding fresh "discrete Laplace" noise to the corresponding actual degree. If the round occurs during phase ϕ, then the server replies with the set of all vertices with noisy degree at most $(2+\eta)^\phi$ (in that round), which are to be deleted, and estimates the coreness of each such vertex as $(2+\eta)^{\phi-1}$. The surviving vertices update their degrees and continue to the next round.

This is the motivation for having low-round algorithms.

It can be shown that, if the standard deviation of each noisy degree is $O(T/\varepsilon)$, then the transcript of each round is ε/T-edge differentially private. Therefore, by composition, the entire transcript is ε-edge differentially private. The accuracy of the estimates follows from the following lemma, using the fact that each noisy degree differs from its corresponding actual degree by $O(\varepsilon^{-1} T \log n)$ with probability $1 - n^{-\Omega(1)}$. This lemma is implicitly proved by Dhulipala et al. [20], so we omit it.

Lemma 9.7.1 *If every noisy degree differs from its corresponding actual degree by at most α, then for every vertex $v \in V$ with actual coreness $k(v)$,*

- *the estimated coreness $\tilde{k}(v)$ of v satisfies*

$$k(v) - O(\alpha) \leq \tilde{k}(v) \leq (2+\eta)k(v) + O(\alpha), \quad \text{and}$$

- *every vertex in $G[U]$ has (induced) degree at least $\tilde{k}(v)/(2+\eta) - O(\alpha)$, where U is the set of all vertices u with estimated coreness $\tilde{k}(u) \geq \tilde{k}(v)$.*

Similar to Section 9.4, we observe that it is possible to obtain smaller additive error if each vertex v generates a noisy initial degree using Laplace noise, and then generates its later noisy degrees using a private continual counting mechanism, $\mathscr{C}_v$, which tracks the absolute change in its degree in each round. In particular, since Observation 9.4.1 also holds in this setting, to ensure that the transcript is ε-edge differentially private, it suffices to use $\mathsf{Lap}(\varepsilon/4)$ noise and for each $\mathscr{C}_v$ to be $\varepsilon/2$-differentially private. The privacy proof is nearly verbatim the same as Lemma 9.4.2, so we omit it.

Each vertex makes $T = O(\log_{1+\eta} n \log n)$ insertions. Taking $\beta = (T \cdot n^{\Omega(1)})^{-1}$, the binary tree mechanism guarantees an error bound of

$$O(\varepsilon^{-1} \log n \log \log_{1+\eta} n)$$

by Theorem 3.2.2. Combining this with Lemma 9.7.1 immediately yields

Theorem 9.4.5 *Given $\varepsilon, \eta > 0$, there is a local ε-edge differentially private mechanism $\mathscr{M}$ that returns an estimate of the coreness of each vertex in an n-vertex graph $G = (V, E)$ such that:*

- *$\mathscr{M}$ runs for $O(\log_{1+\eta} n \log n)$ rounds in any execution, and*
- *for each vertex $v \in V$ with actual coreness $k(v)$, the estimated coreness $\tilde{k}(v)$ of v satisfies:*

$$k(v) - \alpha \leq \tilde{k}(v) \leq (2+\eta)k(v) + \alpha\,,$$

where $\alpha = O(\varepsilon^{-1} \log n \log \log_{1+\eta} n)$ with probability $1 - n^{-\Omega(1)}$.

9.7.2. Approximate Densest Subgraph

It can be shown that if $G[U^*]$ is a densest subgraph of G, then the (induced) degree of every vertex in $G[U^*]$ is at least its density, ρ^*.[7] Hence, $G[U^*]$ is contained in the ρ^*-core of G and ρ^* is at most the maximum coreness, k^*.

7: If some vertex has degree less than ρ^*, then removing it yields a subgraph with higher density, which is impossible.

Lemma 9.7.2 (Folklore) *Given a graph G, if ρ^* is maximum density of any subgraph of G and k^* is the maximum coreness of any $v \in G$, then $\rho^* \leq k^*$.*

This leads to a simple 2-approximation algorithm for densest subgraph. Let H be the k^*-core of G. Since every vertex in H has (induced) degree at least k^*, there are at least $k^*|V(H)|/2$ edges in H. Therefore, $\rho(H)$ satisfies

$$\rho(H) = \frac{|E(H)|}{|V(H)|} \geq \frac{k^*}{2} \geq \frac{\rho^*}{2}$$

and returning H yields a 2-approximate densest subgraph for G.

Dhulipala et al. [20, 150] leveraged the preceding observation to develop differentially private 2-approximate and $(4+\eta)$-approximate solutions to the

densest subgraph problem in the centralized and local model, respectively. Specifically, their mechanisms return a set of vertices U and U' such that the density of $G[U]$ and $G[U']$ is at least $\rho^*/2$ and $\rho^*/(4+\eta)$, respectively. Their procedures may be summarized as follows.

(1) Compute an estimate $\tilde{k}(v)$ of the coreness of every vertex v in G.
(2) Return $\tilde{U}^* = \{v \in V : \tilde{k}(v) = \tilde{k}^*\}$, where $\tilde{k}^* = \max\{\tilde{k}(v) : v \in V\}$.

Since the second step is simply post-processing, the entire procedure has the same differential privacy guarantees as the coreness estimation procedure employed in the first step.

This lemma gives sufficient conditions for the accuracy of the estimation to scale with the accuracy of the underlying coreness estimation procedure. Roughly speaking, Dhulipala et al. [150] assumes their coreness estimation procedure satisfies the second property without proof, while Dhulipala et al. [20] implicitly prove it.

Lemma 9.7.3 *Let $\gamma \geq 1$ be a constant. Suppose that for each vertex $v \in V$ with actual coreness $k(v)$:*

- *the estimated coreness $\tilde{k}(v)$ of v satisfies $k(v) - \alpha \leq \tilde{k}(v) \leq \gamma k(v) + \alpha$ and*
- *every vertex in $G[U]$ has (induced) degree at least $\tilde{k}(v)/\gamma - \alpha$, where U is the set of all vertices u with estimated coreness $\tilde{k}(u) \geq \tilde{k}(v)$.*

Then the density of $G[\tilde{U}^*]$ is at least $\rho^*/2\gamma - O(\alpha)$.

Proof. By the first assumption,

$$\tilde{k}^* = \max\{\tilde{k}(v) \mid v \in V\} \geq \max\{k(v) - \alpha \mid v \in V\} = k^* - \alpha.$$

We have that

$$\begin{aligned}
\frac{|E(G[\tilde{U}^*])|}{|V(G[\tilde{U}^*])|} &\geq \sum_{v \in \tilde{U}^*} \frac{1}{2} \cdot \deg_{\tilde{U}^*}(v) \\
&\geq \frac{1}{2} \cdot \left(\frac{\tilde{k}^*}{\gamma} - \alpha\right) && \text{(by the second assumption)} \\
&\geq \frac{1}{2} \cdot \left(\frac{k^*}{\gamma} - \left(1 + \frac{1}{\gamma}\right)\alpha\right) && \text{(by the first assumption)} \\
&\geq \frac{\rho^*}{2\gamma} - O(\alpha) && \text{(by Lemma 9.7.2)}
\end{aligned}$$

as required. □

By employing our coreness estimation procedures from Sections 9.4 and 9.7.1, which both satisfy the conditions of Lemma 9.7.3 (see Lemmas 9.4.4 and 9.7.1), we immediately obtain the following.

Theorem 9.4.6 *Given $\varepsilon, \eta > 0$, there are local ε-edge differentially private mechanisms $\mathscr{M}$ and $\mathscr{M}'$ that return subsets of vertices U and U' from a given n-vertex graph G, respectively, such that:*

- *$\mathscr{M}$ and $\mathscr{M}'$ run for at most n and $O(\log_{1+\eta} n \log\log n)$ rounds, respectively, in any execution.*
- *With probability $1 - n^{-\Omega(1)}$, the density of $G[U]$ and $G[U']$ is at least $\rho^*/2 - \alpha$ and $\rho^*/(4+\eta) - \alpha'$, respectively, where ρ^* is the maximum density of any subgraph of G, $\alpha = O(\varepsilon^{-1} \log n \log \Delta)$, $\alpha' = O(\varepsilon^{-1} \log_{1+\eta} n \log\log n)$, and Δ is the maximum degree of G.*

9.8. Inner Product Queries

De [156] proved a lower bound on the additive error of any differentially private mechanism that answers $m = O(n)$ random inner product queries in $\{-1,1\}^n$ on a secret dataset in $\{0,1\}^n$. Specifically, the lower bound says that if the error is $O(\sqrt{n})$ on a $(\frac{1}{2}+\eta)$-fraction of the responses with probability $\Omega(\sqrt{\delta})$, then the mechanism is not (ε,δ)-differentially private.

Theorem 9.8.1 (Theorem 4.1 of [156]) *Given $n \in \mathbb{N}$, $\varepsilon > 0$, and $\delta \in [0, 0.05]$, there exists positive constants α, γ, and $\eta < 1/2$ such that any mechanism $\mathscr{M}$ that answers $m = \alpha n$ random inner product queries $Q^{(1)}, \dots, Q^{(m)} \in \{-1,1\}^n$ on a secret dataset $X \in \{0,1\}^n$ satisfying*

$$\Pr_{\mathscr{M}, Q^{(1)}, \dots, Q^{(m)}} \left[\Pr_{i \in [m]} \left[|\mathscr{M}(X)_i - \langle X, Q^{(i)} \rangle| \le \gamma \sqrt{n} \right] \ge \frac{1}{2} + \eta \right] \ge 3\sqrt{\delta}$$

where $\mathscr{M}(X)_i$ denotes the response of $\mathscr{M}$ on query $Q^{(i)}$, is not (ε,δ)-differentially private.

An alternate proof is by calculating the discrepancy of a random query matrix, and can even be used to obtain a single query matrix with the same property.

To obtain the lower bound on inner product queries in $\{0,1\}^n$ that we desire, we show that any ε-differentially private mechanism $\tilde{\mathscr{M}}$ that answers m random inner product queries in $\{0,1\}^n$ can be converted into a 2ε-differentially private mechanism $\mathscr{M}$ that answers m random inner product queries in $\{-1,1\}^n$. The idea is to observe that, for any $Q \in \{-1,1\}^n$, we may write

This is just shifting the basis, as shown below.

$$\langle Q, X \rangle = 2\langle \tilde{Q}, X \rangle - \langle \mathbf{1}, X \rangle,$$

where $\tilde{Q} \in \{0,1\}^n$ is such that $\tilde{Q}_i = (\tilde{Q}_i + 1)/2$ and $\mathbf{1} \in \{1\}^n$ is the all-ones vector. Using this, we can simulate $\tilde{\mathscr{M}}$ on X, scale each response by a factor of 2, and then subtract a noisy version of $\langle \mathbf{1}, X \rangle$ from each response. More precisely, to answer m random inner product queries in $\{-1,1\}^n$ on $X \in \{0,1\}^n$, $\mathscr{M}$ does the following:

1. Generate $y \sim \mathrm{Lap}(1/\varepsilon)$ and release $\tilde{x} = \langle \mathbf{1}, X \rangle + y$.
2. Run $\tilde{\mathscr{M}}$ on X to obtain responses $(r_1, r_2, \dots, r_m)$.
3. Return $(2r_1 - \tilde{x}, 2r_2 - \tilde{x}, \dots, 2r_m - \tilde{x})$.

The Laplace noise causes $\Omega(\log n)$ error, but that is dwarfed by the $\Omega(\sqrt{n})$ error from inner product queries.

The first step is ε-differentially private by Fact 3.2.1. The second step is ε-differentially private by assumption. Therefore, by composition, the first two steps are 2ε-differentially private. The last step is simply post-processing.

Notice that since $\mathscr{M}$ answers m random inner product queries $\tilde{Q}^{(1)}, \dots, \tilde{Q}^{(m)} \in \{0,1\}^n$ on X, the vectors $Q^{(1)}, \dots, Q^{(m)} \in \{-1,1\}^n$, where $Q_i^{(j)} = 2\tilde{Q}_i^{(j)} + 1$ are random. Furthermore, the error of the j-th answer is

$$\begin{aligned} &|(2r_j - \tilde{x}) - \langle Q^{(j)}, X \rangle| \\ &\le |2r_j - 2\langle \tilde{Q}^{(j)}, X \rangle| + |2\langle \tilde{Q}^{(j)}, X \rangle - \langle \mathbf{1}, X \rangle - \langle Q^{(j)}, X \rangle| + |y| \\ &= 2|r_j - \langle \tilde{Q}^{(j)}, X \rangle| + |y|. \end{aligned}$$

By Fact 3.2.2, $|y| \le O(\varepsilon^{-1} \log n)$ with high probability. Therefore, if the error of j-th answer of $\tilde{\mathscr{M}}$ is $o(\sqrt{n})$, then so is the error of the j-th answer of $\mathscr{M}$, assuming ε is a constant. This gives us Theorem 9.5.3.

10. Conclusion

Isidora, therefore, is the city of his dreams: with one difference. The dreamed-of city contained him as a young man; he arrives at Isidora in his old age. In the square there is the wall where the old men sit and watch the young go by; he is seated in a row with them. Desires are already memories.

Italo Calvino, *Invisible Cities*

To conclude, we discuss a few intriguing open problems related to the questions studied in this thesis, and then discuss follow up work that improved the results presented in the thesis.

10.1. Open Problems

Incremental Maximum Flow. In Chapter IncrementalFlow, we gave a combinatorial algorithm for incremental $(1-\varepsilon)$-approximate maximum flow on an undirected, uncapacitated graph. A natural follow up question is

> Can you obtain a combinatorial algorithm for incremental *exact* maximum flow with polylogarithmic update time on dense graphs?

The question is restricted to dense graphs since it seems easier to obtain such combinatorial results on dense graphs.

Electrical Oblivious Routing. In Chapter ElectricalRouting, we showed how to use $O(\sqrt{m})$ electrical routings to obtain competitive ratio $O(\log^2 n)$. There are two natural questions:

> What is the best competitive ratio achievable when using a single electrical routing?

If this can be shown to be $O(\alpha_{\text{LOCAL}})$, then it gives an $O(m)$ space oblivious routing with polylogarithmic competitive ratio.

> Is the value of electrical flow localization either $\alpha_{\text{LOCAL}} = O(\log n)$ or $\Omega(\log^2 n)$?

If this is $O(\log n)$, then electrical flows localize optimally, since oblivious routing has $\Omega(\log n)$ competitive ratio on grids.

Histogram. In Chapter Histogram, we gave a mechanism that separated the dependence of d and $\log T$ when the maximum column sum was small. The general question, however, is still very interesting and unresolved.

> Can you either:
> obtain a histogram algorithm with error $\widetilde{O}(d/\varepsilon) + O(\text{polylog}(T)/\varepsilon)$, or show that every such algorithm must admit $\widetilde{\Omega}(d\,\text{polylog}(T)/\varepsilon)$ error?

We showed that the second case is true for independently private algorithms, so an algorithm which takes the interaction of the columns into account would be novel.

Counting Distinct Elements. In Chapter CountDistinct, we gave a mechanism that had tight bounds in the item-level setting. The following question in the event-level setting is still interesting.

In ongoing work, we show that resolving this question would result in tight bounds for various other continual observation problems as well.

What is the correct bound in the event-level setting in terms of T, since the lower bound is $\sqrt[4]{T}$ and the upper bound is $\sqrt[3]{T}$?

Private kCore. In the ESA 2025 paper, we show that the Multi-Dimensional AboveThreshold mechanism achieves error $O(\log n/\varepsilon)$ for k-core, which closes this line of work. The following question is still curiously open.

Since high density subgraphs need a lot of edge changes before their density is affected, neighboring graphs have nearly the same density, which just might admit high-accuracy mechanisms.

What is the additive error under local edge differential privacy for single-round densest subgraph?

10.2. Follow-Up Work

[160]: Abboud et al. (2024), "Worst-Case to Expander-Case Reductions: Derandomized and Generalized"

Dynamic Lower Bounds. Abboud and Wallheimer [160] reduce worst-case hardness of problems to hardness on expanders, and in the process give alternate and stronger versions of our reductions to expander graphs.

[161]: Henzinger et al. (2024), "Concurrent Composition for Continual Mechanisms"

Histogram. Henzinger, Safavi, and Vadhan [161] have given a simpler proof of the privacy of the first mechanism, namely the interaction between the partitioning and the histogram mechanism using new concurrent composition theorems for continual observation.

[162]: Raskhodnikova et al. (2025), "Fully Dynamic Algorithms for Graph Databases with Edge Differential Privacy"

Counting Distinct Elements. Raskhodnikova and Steiner [162], among other results, also consider event-level output-determined algorithms, and give tight bounds for such algorithms for a variety of graph problems.

Bibliography

[1] E. W. Dijkstra.
A note on two problems in connexion with graphs.
Numerische Mathematik 1.1 (Dec. 1959).

[2] Ellis L. Johnson.
On shortest paths and sorting.
ACM Annual Conference 1972.

[3] Michael L. Fredman and Robert Endre Tarjan.
Fibonacci heaps and their uses in improved network optimization algorithms.
J. ACM 34.3 (1987).

[4] L. R. Ford and D. R. Fulkerson.
Maximal Flow Through a Network.
Canadian Journal of Mathematics 8 (1956).

[5] Daniel Dominic Sleator and Robert Endre Tarjan.
A Data Structure for Dynamic Trees.
J. Comput. Syst. Sci. 26.3 (1983).

[6] 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.
Foundations of Computer Science (FOCS) 2022.

[7] Latanya Sweeney.
Weaving technology and policy together to maintain confidentiality.
J. Law Med. Ethics 25.2-3 (1997).

[8] Arvind Narayanan and Vitaly Shmatikov.
Robust De-anonymization of Large Sparse Datasets.
Symposium on Security and Privacy (S&P) 2008.

[9] Anthony Tockar.
Riding with the Stars: Passenger Privacy in the NYC Taxicab Dataset.
2014 (last accessed 10.2025).

[10] Latanya Sweeney.
k-anonymity: A Model for Protecting Privacy.
International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10.05 (2002).

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith.
Calibrating Noise to Sensitivity in Private Data Analysis.
Theory of Cryptography Conference (TCC) 2006.

[12] John M. Abowd et al..
The modernization of statistical disclosure limitation at the US Census Bureau.
Census Working Papers (2020).

[13] Branislav Stojkovic et al..
Applied Federated Learning: Architectural Design for Robust and Efficient Learning in Privacy Aware Settings.
CoRR abs/2206.00807 (2022).

[14] Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer.
Dynamic algorithms against an adaptive adversary: generic constructions and lower bounds.
Symposium on Theory of Computing (STOC) 2022.

[15] Harald Räcke.
Minimizing Congestion in General Networks.
Foundations of Computer Science (FOCS) 2002.

[16] Thatchaphol Saranurak and Di Wang.
Expander Decomposition and Pruning: Faster, Stronger, and Simpler.
Symposium on Discrete Algorithms (SODA) 2019.

[17] David R. Karger and Matthew S. Levine.
Fast Augmenting Paths by Random Sampling from Residual Graphs.
SIAM J. Comput. 44.2 (2015).

[18] Cynthia Dwork, Moni Naor, Omer Reingold, and Guy N. Rothblum.
Pure Differential Privacy for Rectangle Queries via Private Partitions.
Advances in Cryptology (ASIACRYPT) 2015.

[19] Palak Jain, Iden Kalemaj, Sofya Raskhodnikova, Satchit Sivakumar, and Adam Smith.
Counting Distinct Elements in the Turnstile Model with Differential Privacy under Continual Observation.
Neural Information Processing Systems (NeurIPS) 2023.

[20] Laxman Dhulipala, Quanquan C. Liu, Sofya Raskhodnikova, Jessica Shi, Julian Shun, and Shangdi Yu.
Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs.
Foundations of Computer Science (FOCS) 2022.

[21] Cynthia Dwork and Aaron Roth.
The Algorithmic Foundations of Differential Privacy.
Found. Trends Theor. Comput. Sci. 9.3-4 (2014).

[22] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum.
Differential privacy under continual observation.
Symposium on Theory of Computing (STOC) 2010.

[23] T.-H. Hubert Chan, Elaine Shi, and Dawn Song.
Private and Continual Release of Statistics.
ACM Trans. Inf. Syst. Secur. 14.3 (2011).

[24] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan.
On the complexity of differentially private data release: efficient algorithms and hardness results.
Symposium on Theory of Computing (STOC) 2009.

[25] Min Lyu, Dong Su, and Ninghui Li.
Understanding the Sparse Vector Technique for Differential Privacy.
Proc. VLDB Endow. 10.6 (2017).

[26] Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, and MR Reddy.
Applications of network optimization.
Handbooks in Operations Research and Management Science 7 (1995).

[27] Sanjeev Arora, Elad Hazan, and Satyen Kale.
The Multiplicative Weights Update Method: a Meta-Algorithm and Applications.
Theory Comput. 8.1 (2012).

[28] Jonah Sherman.
Breaking the Multicommodity Flow Barrier for O(vlog n)-Approximations to Sparsest Cut.
Foundations of Computer Science (FOCS) 2009.

[29] Andrew V. Goldberg and Satish Rao.
Beyond the Flow Decomposition Barrier.
J. ACM 45.5 (1998).

[30] Shang-Hua Teng.
The Laplacian Paradigm: Emerging Algorithms for Massive Graphs.
Theory and Applications of Models of Computation (TAMC) 2010.

[31] Jonah Sherman.
Nearly Maximum Flows in Nearly Linear Time.
Foundations of Computer Science (FOCS) 2013.

[32] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford.
An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations.
Symposium on Discrete Algorithms (SODA) 2014.

[33] Richard Peng.
Approximate Undirected Maximum Flows in $O(m\text{polylog}(n))$ Time.
Symposium on Discrete Algorithms (SODA) 2016.

[34] Jonah Sherman.
Area-convexity, l_∞ regularization, and undirected multicommodity flow.
Symposium on Theory of Computing (STOC) 2017.

[35] Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg.
Almost-Linear Time Algorithms for Incremental Graphs: Cycle Detection, SCCs, s-t Shortest Path, and Minimum-Cost Flow.
Symposium on Theory of Computing (STOC) 2024.

[36] Søren Dahlgaard.
On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter.
International Colloqium on Automata, Languages, and Programming (ICALP) 2016.

[37] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak.
Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture.
Symposium on Theory of Computing (STOC) 2015.

[38] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak.
Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers.
Foundations of Computer Science (FOCS) 2020.

[39] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan.
The Expander Hierarchy and its Applications to Dynamic Graph Algorithms.
Symposium on Discrete Algorithms (SODA) 2021.

[40] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva.
Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality.
Foundations of Computer Science (FOCS) 2024.

[41] Monika Rauch Henzinger.
A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity.
J. Algorithms 24.1 (1997).

[42] Manoj Gupta and Shahbaz Khan.
Simple dynamic algorithms for Maximal Independent Set, Maximum Flow and Maximum Matching.
Symposium on Simplicity in Algorithms (SOSA) 2021.

[43] Gramoz Goranci and Monika Henzinger.
Efficient Data Structures for Incremental Exact and Approximate Maximum Flow.
International Colloqium on Automata, Languages, and Programming (ICALP) 2023.

[44] Jan van den Brand, Yang P. Liu, and Aaron Sidford.
Dynamic Maxflow via Dynamic Interior Point Methods.
Symposium on Theory of Computing (STOC) 2023.

[45] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford.
Incremental Approximate Maximum Flow on Undirected Graphs in Subpolynomial Update Time.
Symposium on Discrete Algorithms (SODA) 2024.

[46] Hiroshi Nagamochi and Toshihide Ibaraki.
A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph.
Algorithmica 7.5&6 (1992).

[47] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi.
A General Framework for Graph Sparsification.
SIAM J. Comput. 48.4 (2019).

[48] Alina Ene, Gary L. Miller, Jakub Pachocki, and Aaron Sidford.
Routing under balance.
Symposium on Theory of Computing (STOC) 2016.

[49] Robert Endre Tarjan and Jan van Leeuwen.
Worst-case Analysis of Set Union Algorithms.
J. ACM 31.2 (1984).

[50] Giuseppe F. Italiano.
Amortized Efficiency of a Path Retrieval Data Structure.
Theor. Comput. Sci. 48.3 (1986).

[51] András A. Benczúr and David R. Karger.
Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs.
SIAM J. Comput. 44.2 (2015).

[52] Daniel A. Spielman and Nikhil Srivastava.
Graph Sparsification by Effective Resistances.
SIAM J. Comput. 40.6 (2011).

[53] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng.
Fully Dynamic Effective Resistances.
CoRR abs/1804.04038 (2018).

[54] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng.
Fully dynamic spectral vertex sparsifiers and applications.
Symposium on Theory of Computing (STOC) 2019.

[55] Yu Gao, Yang P. Liu, and Richard Peng.
Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao.
Foundations of Computer Science (FOCS) 2021.

[56] Ruoxu Cen, Yu Cheng, Debmalya Panigrahi, and Kevin Sun.
Sparsification of Directed Graphs via Cut Balance.
International Colloqium on Automata, Languages, and Programming (ICALP) 2021.

[57] Wolfgang Mader.
A Reduction Method for Edge-Connectivity in Graphs.
Ann. Discrete Math. 3 (1978).

[58] Chris Harrelson, Kirsten Hildrum, and Satish B. Rao.
A Polynomial-time Tree Decomposition to Minimize Congestion.
Symposium on Parallelism in Algorithms and Architectures (SPAA) 2003.

[59] Harald Räcke.
Optimal Hierarchical Decompositions for Congestion Minimization in Networks.
Symposium on Theory of Computing (STOC) 2008.

[60] David Applegate and Edith Cohen.
Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs.
Symposium on Communications Architectures & Protocols (SIGCOMM) 2003.

[61] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé.
Semi-Oblivious Traffic Engineering: The Road Not Taken.
Networked Systems Design and Implementation (NSDI) 2018.

[62] Harald Räcke, Chintan Shah, and Hanjo Täubig.
Computing Cut-Based Hierarchical Decompositions in Almost Linear Time.
Symposium on Discrete Algorithms (SODA) 2014.

[63] Matthias Englert and Harald Räcke.
Oblivious Routing for the Lp-norm.
Foundations of Computer Science (FOCS) 2009.

[64] Gregory Lawler and Hariharan Narayanan.
Mixing Times and lp Bounds for Oblivious Routing.
Meeting on Analytic Algorithmics and Combinatorics (ANALCO) 2009.

[65] Jonathan A. Kelner and Petar Maymounkov.
Electric routing and concurrent flow cutting.
Theor. Comput. Sci. 412.32 (2011).

[66] Aaron Schild, Satish Rao, and Nikhil Srivastava.
Localization of Electrical Flows.
Symposium on Discrete Algorithms (SODA) 2018.

[67] Aaron Sidford and Yin Tat Lee.
Personal communication.
2022.

[68] Sanjeev Arora, Elad Hazan, and Satyen Kale.
The Multiplicative Weights Update Method: a Meta-Algorithm and Applications.
Theory of Computing 8.6 (2012).

[69] Leslie G. Valiant and Gordon J. Brebner.
Universal Schemes for Parallel Communication.
Symposium on Theory of Computing (STOC) 1981.

[70] Prahladh Harsha, Thomas P. Hayes, Hariharan Narayanan, Harald Räcke, and Jaikumar Radhakrishnan.
Minimizing Average Latency in Oblivious Routing.
Symposium on Discrete Algorithms (SODA) 2008.

[71] Aleksander Madry.
Computing Maximum Flow with Augmenting Electrical Flows.
Foundations of Computer Science (FOCS) 2016.

[72] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford.
Faster Maxflow via Improved Dynamic Spectral Vertex Sparsifiers.
Foundations of Computer Science (FOCS) 2022.

[73] Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye.
Nested Dissection Meets IPMs: Planar Min-Cost Flow in Nearly-Linear Time.
Symposium on Discrete Algorithms (SODA) 2022.

[74] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu.
Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs.
Foundations of Computer Science (FOCS) 2020.

[75] Yin Tat Lee, Satish Rao, and Nikhil Srivastava.
A New Approach to Computing Maximum Flows Using Electrical Flows.
Symposium on Theory of Computing (STOC) 2013.

[76] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford.
An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and Its Multicommodity Generalizations.
Symposium on Discrete Algorithms (SODA) 2014.

[77] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng.
Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs.
Symposium on Theory of Computing (STOC) 2011.

[78] Ali Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias.
Robust Routing Using Electrical Flows.
Conference on Advances in Geographic Information Systems (SIGSPATIAL) 2021.

[79] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic.
Hop-Constrained Oblivious Routing.
Symposium on Theory of Computing (STOC) 2021.

[80] Goran Zuzic, Bernhard Haeupler, and Antti Roeyskoe.
Sparse Semi-Oblivious Routing: Few Random Paths Suffice.
Principles of Distributed Computing (PODC) 2023.

[81] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari.
Hop-Constrained Expander Decompositions, Oblivious Routing, and Distributed Universal Optimality.
Symposium on Theory of Computing (STOC) 2022.

[82] Piotr Indyk.
Stable distributions, pseudorandom generators, embeddings, and data stream computation.
J. ACM 53.3 (2006).

[83] Aaron Schild.
An almost-linear time algorithm for uniform random spanning tree generation.
Symposium on Theory of Computing (STOC) 2018.

[84] Jittat Fakcharoenphol, Satish B. Rao, and Kunal Talwar.
A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics.
Symposium on Theory of Computing (STOC) 2003.

[85] Ittai Abraham, Yair Bartal, and Ofer Neiman.
Nearly Tight Low Stretch Spanning Trees.
Foundations of Computer Science (FOCS) 2008.

[86] Huan Li and Aaron Schild.
Spectral Subspace Sparsification.
Foundations of Computer Science (FOCS) 2018.

[87] Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye.
Minor Sparsifiers and the Distributed Laplacian Paradigm.
Foundations of Computer Science (FOCS) 2021.

[88] Richard Peng and Daniel A. Spielman.
An efficient parallel solver for SDD linear systems.
Symposium on Theory of Computing (STOC) 2014.

[89] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman.
Sparsified Cholesky and multigrid solvers for connection laplacians.
Symposium on Theory of Computing (STOC) 2016.

[90] Daniel A. Spielman and Shang-Hua Teng.
Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.
Symposium on Theory of Computing (STOC) 2004.

[91] Arun Jambulapati and Aaron Sidford.
Ultrasparse Ultrasparsifiers and Faster Laplacian System Solvers.
Symposium on Discrete Algorithms (SODA) 2021.

[92] Amir Abboud and Søren Dahlgaard.
Popular Conjectures as a Barrier for Dynamic Planar Graph Algorithms.
Foundations of Computer Science (FOCS) 2016.

[93] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak.
Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture.
Symposium on Theory of Computing (STOC) 2015.

[94] Sairam Subramanian.
A Fully Dynamic Data Structure for Reachability in Planar Digraphs.
European Symposium on Algorithms (ESA) 1993.

[95] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian.
Faster Shortest-Path Algorithms for Planar Graphs.
J. Comput. Syst. Sci. 55.1 (1997).

[96] P. N. Klein and S. Subramanian.
A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs.
Algorithmica 22.3 (1998).

[97] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen.
Improved algorithms for min cut and max flow in undirected planar graphs.
Symposium on Theory of Computing (STOC) 2011.

[98] Ittai Abraham, Shiri Chechik, and Cyril Gavoille.
Fully Dynamic Approximate Distance Oracles for Planar Graphs via Forbidden-Set Distance Labels.
Symposium on Theory of Computing (STOC) 2012.

[99] Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck.
On Dynamic Approximate Shortest Paths for Planar Graphs with Worst-Case Costs.
Symposium on Discrete Algorithms (SODA) 2016.

[100] Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski.
Decremental single-source reachability in planar digraphs.
Symposium on Theory of Computing (STOC) 2017.

[101] Panagiotis Charalampopoulos and Adam Karczmarz.
Single-source shortest paths and strong connectivity in dynamic planar graphs.
J. Comput. Syst. Sci. 124 (2022).

[102] Manoj Gupta and Richard Peng.
Fully Dynamic (1+ϵ)-Approximate Matchings.
Foundations of Computer Science (FOCS) 2013.

[103] Monika Henzinger, Andrea Lincoln, and Barna Saha.
The Complexity of Average-Case Dynamic Subgraph Counting.
Symposium on Discrete Algorithms (SODA) 2022.

[104] Wenyu Jin and Xiaorui Sun.
Fully Dynamic c-Edge Connectivity in Subpolynomial Time.
CoRR abs/2004.07650 (2020).

[105] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan.
The Expander Hierarchy and its Applications to Dynamic Graph Algorithms.
Symposium on Discrete Algorithms (SODA) 2021.

[106] Julia Chuzhoy and Thatchaphol Saranurak.
Deterministic Algorithms for Decremental Shortest Paths via Layered Core Decomposition.
Symposium on Discrete Algorithms (SODA) 2021.

[107] Julia Chuzhoy.
Decremental all-pairs shortest paths in deterministic near-linear time.
Symposium on Theory of Computing (STOC) 2021.

[108] Fan Chung Graham.
Large Dynamic Graphs: What Can Researchers Learn from Them?.
SIAM News 37.3 (2004).

[109] Saurabh Sawlani and Junxing Wang.
Near-Optimal Fully Dynamic Densest Subgraph.
Symposium on Theory of Computing (STOC) 2020.

[110] Gramoz Goranci, Monika Henzinger, and Pan Peng.
Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs.
European Symposium on Algorithms (ESA) 2018.

[111] David Peleg and Shay Solomon.
Dynamic (1 + ϵ)-Approximate Matchings: A Density-Sensitive Approach.
Symposium on Discrete Algorithms (SODA) 2016.

[112] Søren Dahlgaard.
On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter.
CoRR abs/1602.06705 (2016).

[113] Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein.
Algorithms and Hardness for Diameter in Dynamic Graphs.
International Colloquium on Automata, Languages, and Programming (ICALP) 2019.

[114] Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein.
New Techniques and Fine-Grained Hardness for Dynamic Near-Additive Spanners.
Symposium on Discrete Algorithms (SODA) 2021.

[115] William Aiello, Fan Chung Graham, and Linyuan Lu.
A Random Graph Model for Power Law Graphs.
Exp. Math. 10.1 (2001).

[116] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu.
Practical and Private (Deep) Learning Without Sampling or Shuffling.
International Conference on Machine Learning (ICML) 2021.

[117] Qinzi Zhang, Hoang Tran, and Ashok Cutkosky.
Differentially Private Online-to-batch for Smooth Losses.
Neural Information Processing Systems (NeurIPS) 2022.

[118] Sergey Denisov, H. Brendan McMahan, John Rush, Adam D. Smith, and Abhradeep Guha Thakurta.
Improved Differential Privacy for SGD via Optimal Private Linear Operators on Adaptive Streams.
Neural Information Processing Systems (NeurIPS) 2022.

[119] Christopher A. Choquette-Choo, Hugh Brendan McMahan, J. Keith Rush, and Abhradeep Guha Thakurta.
Multi-Epoch Matrix Factorization Mechanisms for Private Machine Learning.
International Conference on Machine Learning (ICML) 2023.

[120] Hilal Asi, Vitaly Feldman, Tomer Koren, and Kunal Talwar.
Near-Optimal Algorithms for Private Online Optimization in the Realizable Regime.
International Conference on Machine Learning (ICML) 2023.

[121] Christopher A. Choquette-Choo, Arun Ganesh, Ryan McKenna, H. Brendan McMahan, John Rush, Abhradeep Guha Thakurta, and Zheng Xu.
(Amplified) Banded Matrix Factorization: A unified approach to private training.
Neural Information Processing Systems (NeurIPS) 2023.

[122] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A. Choquette-Choo, Peter Kairouz, H. Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang.
Federated Learning of Gboard Language Models with Differential Privacy.
Meeting of the Association for Computational Linguistics (ACL): Industry Track 2023.

[123] Hendrik Fichtenberger, Monika Henzinger, and Jalaj Upadhyay.
Constant Matters: Fine-grained Error Bound on Differentially Private Continual Observation.
International Conference on Machine Learning (ICML) 2023.

[124] Monika Henzinger, Jalaj Upadhyay, and Sarvagya Upadhyay.
Almost Tight Error Bounds on Differentially Private Continual Counting.
Symposium on Discrete Algorithms (SODA) 2023.

[125] Joel Daniel Andersson and Rasmus Pagh.
A Smooth Binary Mechanism for Efficient Private Continual Observation.
Neural Information Processing Systems (NeurIPS) 2023.

[126] Monika Henzinger and Jalaj Upadhyay.
Improved Differentially Private Continual Observation Using Group Algebra.
Symposium on Discrete Algorithms (SODA) 2025.

[127] Joel Daniel Andersson, Rasmus Pagh, Teresa Anna Steiner, and Sahel Torkamani.
Count on Your Elders: Laplace vs Gaussian Noise.
Foundations of Responsible Computing (FORC) 2025.

[128] Palak Jain, Sofya Raskhodnikova, Satchit Sivakumar, and Adam D. Smith.
The Price of Differential Privacy under Continual Observation.
International Conference on Machine Learning (ICML) 2023.

[129] Hendrik Fichtenberger, Monika Henzinger, and Lara Ost.
Differentially Private Algorithms for Graphs Under Continual Observation.
European Symposium on Algorithms (ESA) 2021.

[130] Max Dupré la Tour, Monika Henzinger, and David Saulpic.
Making Old Things New: A Unified Algorithm for Differentially Private Clustering.
International Conference on Machine Learning (ICML) 2024.

[131] Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, and Uri Stemmer.
Lower Bounds for Differential Privacy Under Continual Observation and Online Threshold Queries.
Conference on Learning Theory (COLT) 2024.

[132] Yuan Qiu and Ke Yi.
Differential Privacy on Dynamic Data.
CoRR abs/2209.01387 (2022).

[133] Samuel Haney, Michael Shoemate, Grace Tian, Salil P. Vadhan, Andrew Vyrros, Vicki Xu, and Wanrong Zhang.
Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters.
Conference on Computer and Communications Security (CCS) 2023.

[134] Moritz Hardt and Kunal Talwar.
On the geometry of differential privacy.
Symposium on Theory of Computing (STOC) 2010.

[135] Salil P. Vadhan and Tianhao Wang.
Concurrent Composition of Differential Privacy.
Theory of Cryptography Conference (TCC) 2021.

[136] Patricia Guerra-Balboa, Àlex Miranda-Pascual, Javier Parra-Arnau, and Thorsten Strufe.
Composition in Differential Privacy for General Granularity Notions.
Computer Security Foundations Symposium (CSF) 2024.

[137] Philippe Flajolet and G. Nigel Martin.
Probabilistic Counting Algorithms for Data Base Applications.
J. Comput. Syst. Sci. 31.2 (1985).

[138] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier.
HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm.
Conference on Analysis of Algorithms (AofA) 2007.

[139] Daniel M. Kane, Jelani Nelson, and David P. Woodruff.
An optimal algorithm for the distinct elements problem.
Principles of Database Systems (PODS) 2010.

[140] Matti Karppa and Rasmus Pagh.
HyperLogLogLog: Cardinality Estimation With One Log More.
Conference on Knowledge Discovery and Data Mining (KDD) 2022.

[141] Dingyu Wang and Seth Pettie.
Better Cardinality Estimators for HyperLogLog, PCSA, and Beyond.
Principles of Database Systems (PODS) 2023.

[142] Lotte Weedage, Nelly Litvak, and Clara Stegehuis.
Locating highly connected clusters in large networks with HyperLogLog counters.
J. Complex Networks 9.2 (2021).

[143] Aditya Akella, Ashwin Bharambe, Mike Reiter, and Srinivasan Seshan.
Detecting DDoS attacks on ISP networks.
Workshop on Management and Processing of Data Streams (MPDS) 2003.

[144] Vera Clemens, Lars-Christian Schulz, Marten Gartner, and David Hausheer.
DDoS Detection in P4 Using HYPERLOGLOG and COUNTMIN Sketches.
Network Operations and Management Symposium (NOMS) 2023.

[145] Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft.
Private decayed predicate sums on streams.
International Conference on Database Theory (ICDT) 2013.

[146] Alessandro Epasto, Jieming Mao, Andres Muñoz Medina, Vahab Mirrokni, Sergei Vassilvitskii, and Peilin Zhong.
Differentially Private Continual Releases of Streaming Frequency Moment Estimations.
Innovations in Theoretical Computer Science (ITCS) 2023.

[147] Badih Ghazi, Ravi Kumar, Jelani Nelson, and Pasin Manurangsi.
Private Counting of Distinct and k-Occurring Items in Time Windows.
Innovations in Theoretical Computer Science (ITCS) 2023.

[148] Monika Henzinger, A. R. Sricharan, and Teresa Anna Steiner.
Differentially Private Histogram, Predecessor, and Set Cardinality under Continual Observation.
CoRR abs/2306.10428 (2023).

[149] Mark Bun, Jonathan R. Ullman, and Salil P. Vadhan.
Fingerprinting Codes and the Price of Approximate Differential Privacy.
SIAM J. Comput. 47.5 (2018).

[150] Laxman Dhulipala, George Z. Li, and Quanquan C. Liu.
Near-Optimal Differentially Private k-Core Decomposition.
CoRR abs/2312.07706 (2023).

[151] Dung Nguyen and Anil Vullikanti.
Differentially Private Densest Subgraph Detection.
International Conference on Machine Learning (ICML) 2021.

[152] Alireza Farhadi, MohammadTaghi Hajiaghayi, and Elaine Shi.
Differentially Private Densest Subgraph.
Artificial Intelligence and Statistics (AISTATS) 2022.

[153] Michael Dinitz, Satyen Kale, Silvio Lattanzi, and Sergei Vassilvitskii.
Almost Tight Bounds for Differentially Private Densest Subgraph.
Symposium on Discrete Algorithms (SODA) 2025.

[154] Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam D. Smith.
Triangle Counting with Local Edge Differential Privacy.
International Colloquium on Automata, Languages, and Programming (ICALP) 2023.

[155] Matthew Joseph, Jieming Mao, Seth Neel, and Aaron Roth.
The Role of Interactivity in Local Differential Privacy.
Foundations of Computer Science (FOCS) 2019.

[156] Anindya De.
Lower Bounds in Differential Privacy.
Theory of Cryptography Conference (TCC) 2012.

[157] Amos Beimel, Kobbi Nissim, and Eran Omri.
Distributed Private Data Analysis: Simultaneously Solving How and What.
Advances in Cryptology (CRYPTO) 2008.

[158] David W. Matula and Leland L. Beck.
Smallest-Last Ordering and clustering and Graph Coloring Algorithms.
J. ACM 30.3 (1983).

[159] Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, and Julian Shun.
Parallel Batch-Dynamic Algorithms for k-Core Decomposition and Related Graph Problems.
Symposium on Parallelism in Algorithms and Architectures (SPAA) 2022.

[160] Amir Abboud and Nathan Wallheimer.
Worst-Case to Expander-Case Reductions: Derandomized and Generalized.
European Symposium on Algorithms (ESA) 2024.

[161] Monika Henzinger, Roodabeh Safavi, and Salil P. Vadhan.
Concurrent Composition for Continual Mechanisms.
CoRR abs/2411.03299 (2024).

[162] Sofya Raskhodnikova and Teresa Anna Steiner.
Fully Dynamic Algorithms for Graph Databases with Edge Differential Privacy.
Principles of Database Systems (PODS) 2025.