WARWICK
THE UNIVERSITY OF WARWICK

Finding Structure in Data Streams:
Correlations, Independent Sets, and Matchings

by

Jacques Dark

Thesis

Submitted to the University of Warwick
for the degree of

Doctor of Philosophy

Department of Computer Science
January 2020

Contents

Acknowledgements

First and foremost I would like to thank my supervisor, Graham Cormode, without whom I would not have discovered the fascinating world of sketching, streaming, and communication complexity.

I am also extremely grateful to Christian Konrad. Discussions with Christian in my second year first sparked my interest in graph streaming lower bounds, eventually leading to the development of all the work in Chapters 4 and 5. Christian has continued to be a great source of interesting problems and ideas.

I would like to thank the various members of the University of Warwick streaming and privacy research group, particularly Charlie Dickens - who I have had many helpful conversations with.

Financially, this work was supported by a Microsoft EMEA PhD Scholarship, a stipend top-up from the Alan Turing Institute (for my placement year with them in London), and the various opportunities provided by the EPSRC-funded Warwick Institute for the Science of Cities.

And of course, last but not least, I would like to thank all the people who have supported me directly and indirectly throughout the years, and especially as I was writing up. Thank you so much Mirabelle Knowles, John Humphreys, Nigel Heathcote, Matthew Green, and Chloë Simons; my parents - Paul Dark and Marie Rölli; and the many more who could be listed here.

Declaration

This thesis is an original presentation based primarily on my published works:

- Chapters 1 and 2 summarise the important models, techniques, and results from the field of streaming algorithms that are necessary to understand the following chapters.
- Chapter 3 is mostly based on "Fast Sketch-based Recovery of Correlation Outlier" [CD18], co-authored with Graham Cormode[1], and presented at the 21st International Conference in Database Theory (ICDT 2018).
- Chapter 4 is mostly based on two papers: "Approximating the Caro-Wei Bound for Independent Sets in Graph Streams" [CDK18], co-authored with Graham Cormode[1] and Christian Konrad[2], and presented at the 5$^{\text{th}}$ International Symposium on Combinatorial Optimization (ISCO 2018); and "Independent Sets in Vertex-Arrival Streams" [CDK19], also co-authored with Graham Cormode[1] and Christian Konrad[2], and presented at the 46$^{\text{th}}$ International Colloquium on Automata, Languages, and Programming (ICALP 2019).
- Chapter 5 is original work co-authored with Christian Konrad[2]. This was unpublished at the time of the original thesis submission, but the results of Section 5.4 were later reformulated and extended into "Optimal Lower Bounds for Matching and Vertex Cover in Dynamic Graph Streams" [DK20], and presented at the 35$^{\text{th}}$ Computational Complexity Conference (CCC 2020).

None of the work presented in this thesis has been submitted for a previous degree at any university, and all of the work was conducted during my period of study at the University of Warwick.

[1] g.cormode@warwick.ac.uk
Department of Computer Science, University of Warwick, Coventry, UK
[2] christian.konrad@bristol.ac.uk
Department of Computer Science, University of Bristol, Bristol, UK

Abstract

The streaming model supposes that, rather than being available all at once, the data is received in a piecemeal fashion. In a world of massive data sets, streaming algorithms give a complementary approach to distributed algorithms: with the data all being available in one place but at different times, rather than at the same time in different places.

We examine three different single-pass streaming problems where existing results show limited feasibility. We consider realistic relaxations or restrictions of these problems which allow for more efficient algorithms.

In the correlation outliers problem, we wish to identify pairs of unusually correlated signals from a streamed matrix of observations. We show that a simple application of existing technique is space-optimal but has slow query time when the outlier threshold is small. We demonstrate how we can achieve faster query times at the cost of storing a larger data summary.

In the maximum independent set problem, we wish to find an edge-less induced subgraph of maximum size. For arbitrary graphs, given as a stream of edges, it is known that no space-efficient algorithm exists. We consider a variant streaming model, where the graph is received vertex by vertex. While we show this model still does not admit efficient algorithms for general graphs, we demonstrate efficient approximation algorithms for various special graph classes.

In the maximum matching problem, we wish to find a disjoint subset of edges of largest possible size. The greedy algorithm gives us an easy 2-approximation for streams of edges, but the problem becomes infeasible to solve if we allow unlimited edge deletions. We consider a model where, instead, a limited number of deletions are allowed. We describe several new approximation algorithms with complexity parameterised by the number of deletions. We also present new techniques which may lead to the development of corresponding tight lower bounds.

1 Introduction

Suppose we have a dataset $\mathcal{D}$ from which we want to compute some function $f(\mathcal{D})$. However, the data is much too large - many times too large to store on our machine. Instead, $\mathcal{D}$ is broken up into n much smaller pieces $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n$, which are provided as input to our machine one-by-one in an arbitrary order.

Can we still compute or estimate $f(\mathcal{D})$ under this circumstance? How restricted can our machine's local storage be (as a function of the input size $|\mathcal{D}|$) before this task becomes impossible?

This is a generalisation of the *data stream model*, which first began appearing in the research literature about 40 years ago, and has received a lot of attention since then. In this thesis, we present new contributions to the understanding of several fundamental problems in the streaming model, as well as new techniques for proving data stream lower bounds.

1.1 Motivation

As almost any article on the topic of data science will loudly tell you, we are in an era of "big data". This term refers not only to the simple fact that bigger and bigger datasets are being seen across all areas of science and industry, but also to the challenges associated with managing and analysing these huge datasets.

The efficient polynomial-time algorithms traditionally taught to undergraduates assume that the full input data is available in memory all at once. However, in the world of big data, the input could simply be too massive for this to be feasible. Instead, it might be distributed across many machines, or held in some slow-to-access data store, or perhaps the data simply never even coexists temporally and is instead constantly being generated, inspected, and then deleted.

In all these various situations the difficulty is the same: we do not have (easy) random access to the data, and we are limited in our local storage.

The data stream model is one way of abstracting these restrictions. We limit ourselves to taking a single uncontrolled pass over the data (i.e. no random access) and we assume the data is much much larger than the available memory. This model manages

to be restrictive enough to be applicable in a wide range of realistic scenarios, while still being powerful enough to allow us to create useful algorithms.

Example: Network Traffic Analysis. The classic example of a data streaming problem is that of a router trying to analyse traffic in a busy network. The router can see the source and destination IP address of all the many packets passing across it, but has relatively limited storage available.

Still, we might still like it to perform some basic statistical analyses of the data, answering questions like: what fraction of the addresses seen today have not been seen before, which addresses are most popular, and have there been any anomalous spikes in activity?

These are exactly data streaming problems.

Example: Database Optimisation. A less direct application is to database query optimisation. When a database receives a new query, it has many different ways it could go about computing the requested output. Each of these different *query plans* would take a different amount of time to complete, so choosing the the right plan is very important.

However, the speed of each plan varies depending on the data stored in the tables and, in particular, on the sizes of the intermediate results generated during the plan's execution. Database management systems use simple heuristics to guess at the size of these intermediates and try to find a good query plan, but improving these estimates can have a dramatic impact on performance [Lei+18].

The question of how to produce good estimates of these intermediate cardinalities with only small overhead is exactly a streaming problem. See, for example, [AMS99; Alo+02] for early work on estimating simple joins and [Ven+15; CY17] for more recent work on multi-joins with filter conditions.

Example: Social Media Graphs. Social media data (from the likes of Facebook) is a good example of a very large, dynamic, graph-like dataset. We can view accounts as graph vertices and use edges to represent the presence of a "friendship" relationship between two accounts. This graph will have billions of nodes and hundreds of billions of edges. Edges and vertices will be added and removed from the graph over time - as accounts are created and deleted, and accounts friend and unfriend each other.

1.2 Topics in Streaming

Forty years on from the earliest works of a streaming flavour, there is a rich and diverse streaming literature. This section represents a brief and non-exhaustive overview of the field, with a particular focus on the areas closest to the work in this thesis.

See the survey of Muthukrishnan [Mut05] for a more detailed overview of the early streaming landscape.

Vector Streams

These are the simplest and most widely studied class of data streams. Broadly speaking, we can group them into three base variants: *insert-once* (or *time-series*) streams, *insert-only* streams, and *turnstile* streams. In each case, the sequence of inputs making up the data stream defines some n-dimensional vector x, from which we wish to compute some $f(x)$.

In an insert-once stream, we simply receive the entries of the vector x directly, although in some arbitrary order. Each stream input (i, x_i) tells us the index i and value x_i of one entry of x (with each entry appearing exactly once) until we have received the entire vector.

The other two models provide x less directly. Each stream item tells us some incremental change (i, Δ) to an entry x_i of x. To find x, you would have to sum all the changes Δ for each x_i. In an insert-only stream, all these changes must be positive $(\Delta > 0)$, while in a turnstile stream they can also be negative.

Further vector stream variants can usually be seen as small modifications of one of the above models. For example, some authors consider turnstile model variants where the final vector x is promised to have all non-negative entries.

Early Works. Even before the streaming model was formalised, there were works with a strong streaming flavour.

Munro and Patterson [MP80] considered what is essentially an insert-once streaming model, motivated by the task of processing data stored on a read-only tape. They studied the space and pass complexity of finding the k^{th}-highest value, and gave optimal space bounds for the problem of finding the median deterministically in one pass or with high probability in two passes.

Flajolet and Martin [FM85] demonstrated a single-pass algorithm for approximately counting the number of distinct elements in a collection of size n using $O(\log n)$ bits.

Frequency Moments. In an influential paper - which would later win them the 2005 Gödel prize for foundational contributions to streaming algorithms - Alon, Matias,

and Szegedy [AMS99] considered the problem of estimating frequency moments of a vector provided as a kind of insert-only data stream (although they also did not yet use the term "stream").

Their stream consisted of a sequence of unit weight increments to the underlying vector. They asked: given one pass over the stream, what is the space required to estimate the p^{th} frequency moment $\|x\|_p^p = \sum_{i=1}^n |x_i|^p$ for $p > 0$? What about the maximum $\|x\|_\infty = \max_{i \in [n]}\{x_i\}$?

They demonstrated sublinear space algorithms for approximating $\|x\|_k^k$ with small relative error (and high probability of success). On the other hand, they showed that approximating $\|x\|_\infty$ to within a constant factor (and with constant probability) requires $\Omega(n)$ bits of space, meaning we may as well keep track of the whole of x.

For the special case of $p = 2$, they proposed a more efficient strategy using only polylog(n) bits of space. By sampling from a 4-wise independent family of vectors with entries ± 1, they described how to randomly project the streamed vector down into a small number of dimensions while approximately preserving the Euclidean norm (with high probability).

Since this strategy only needs to maintain a linear function of the streamed input vector (called a *linear sketch*, or simply a *sketch*), it works just as well in the more general turnstile streaming model.

Later works [Ind06; Li08] generalised this sketching strategy to other $p \in (0, 2]$ through the use of p-stable distributions - giving us polylog(n) space turnstile streaming algorithms for all frequency moments in this range.

New work on lower bounds [Woo04; KNW10b] showed that these sketches are essentially space-optimal for the turnstile streaming model. Although, there was still room to improve the time required to process each stream update [CCFC02; TZ04; NW10; Kan+11].

On the other hand, it was shown [BY+04b] that the best turnstile streaming algorithm requires polynomial space (something like $n^{1-2/p}$ bits) for any $p > 2$. This bound is also known to be tight [IW05; Bhu+06].

Sampling Sampling is an extremely powerful primitive which can be used as a basic building block to implement more complex algorithms and estimate many different functions.

For insert-only streams, we can use reservoir sampling, which has been known since at least the 60s [Knu69, Algorithm R in Section 3.4.2][1].

[1] Credited to Alan G. Waterman

Some 50 years later, Monemizadeh and Woodruff [MW10] described l_p samplers - sketches which allow you to draw samples from the support of the vector with probabilities (approximately) proportional to their l_p weights - making it possible to do sampling over turnstile streams.

These techniques have proven to be a invaluable for developing more complex streaming algorithms, including some of the algorithms in Chapter 5.

Other Vector Problems. Beyond these fundamental tasks, researchers have explored how to compute many different functions on vector streams including: negative frequency moments [BC15], entropy [CCM07], quantiles [Wan+13; KLL16], heavy-hitters [CH08], and inner products [Alo+02].

Thinking more generally, we can ask whether it is possible to design "univeral" algorithms which approximate all (or many) functions simultaneously, allowing the data to be processed without even knowing which function we might want to apply. This has been studied for generalised frequency moments [BOR15], subset norms [BKY20], and symmetric norms [Bła+17].

Matrix Streams

Naturally, as techniques for vector streams matured, researchers began wondering how these ideas could be applied to problems on matrix-like data. For example, the survey of Muthukrishnan [Mut05] noted that relatively little was known about the complexity of solving linear algebra problems in the streaming setting.

For matrices, most of the interest is in insert-once and turnstile streaming models. Like vector streams, in a turnstile matrix stream the stream consists of arbitrary increments and decrements to matrix entries, which sum to give the final input matrix. And in insert-once matrix streams, we simply receive the entries of the final input matrix one-by-one, perhaps in arbitrary order, or perhaps in some structured order (depending on the exact model).

Sketch-and-Solve. By far the most successful approach to building algorithms for matrix streams has been the so-called "sketch-and-solve" paradigm. Roughly speaking, we:

1. Sample a random linear map from a specially chosen family (mapping a high dimensional space into a lower dimensional one).
2. Apply the linear map to the input matrix, compressing it.
3. Solve the original problem on the new compressed matrix.

As we observed with vector streams, the fact that the intermediate algorithm state is a linear function of the input means this applies to the more general turnstile model.

With the right choices of family, this general approach has been used to provide massive speed-up and space saving for many different problems, at the cost of introducing a small random error.

Sarlós [Sar06] was the first to demonstrate how this approach could provide fast data-independent randomised algorithms for various common tasks in linear algebra (such as: computation of matrix products, singular value decompositions, matrix norms, and solutions to linear regressions).

Clarkson and Woodruff [CW09] later provided lower bounds as well as improved upper bounds for some of these problems, and a series of results [RT08; MM13; NN13; BDN15; CW17] provided further improvements to the complexity.

Woodruff wrote an excellent book [Woo14] that covers (in quite a lot of detail) the problems and techniques in this area.

Our work in Section 3.3 follows this rough pattern, with a focus on how to speed up the solve step for a particular correlation detection problem.

Other Work. Beyond the general turnstile model, we can consider more restricted situations. For example, suppose the input matrices are provided as an insert-once stream but either row-by-row or column-by-column. We can hope to leverage this additional structure to design more efficient algorithms.

One deceptively simple approach is to collect a representative sample of the received rows by picking each independently with probability proportional to its Euclidean norm. See [FKV04; RV07] for analysis of this strategy.

Another clever approach due to Ghashami, Liberty, Phillips, and Woodruff [Gha+16] works by processing the arriving rows in batches, periodically computing the singular value decomposition of all stored rows, and then discarding the less important rows.

Graph Streams

More recently, there has been a lot of interest in studying algorithms for streams of graph-like data. The earliest work of this kind is that of Henzinger, Raghavan, and Rajagopalan [HRR98] who considered some path problems on directed graphs.

Since then, there have been many developments. A useful survey by McGregor [McG14] gives good coverage of the first decade and a half.

Models. In the most commonly studied model - the *insert-only edge streaming model* - we receive the edges of the graph one-by-one as the items of the stream. If we further allow edge deletions to occur in the stream, we get the *turnstile edge streaming model*.

Some authors consider *vertex streams*, where the vertices of the graph are the items arriving one-by-one in the stream. There are two major variants: each vertex is received with either its full incidence information (i.e. all the edges in its neighborhood in the full graph) or only its edges to prior vertices in the stream (so that the stream represents an increasing induced subgraph). We will refer to the former variant as the *vertex incidence list model* and the latter as the *explicit vertex model* (to match terminology from [CDK19]).

While most interesting streaming algorithms for vector problems tend to run in $O(\text{polylog}\, n)$ bits of space, this is not sufficient space for computing or estimating many basic graph properties (e.g. connected components). Several authors proposed that the design space of algorithms using $O(\text{polylog}\, n)$ bits *per node* was a reasonable and promising middle ground to study [Mut05; Fei+05], and named this domain the "semi-streaming model". It has proven true that most interesting graph streaming algorithms do lie in this domain; however, in this thesis we will not make a distinction between the semi-streaming model and models that allow more or less space.

Connectivity and Bipartiteness. In insert-only edge streams, a simple disjoint set data structure (slightly augmented) can be used to find the connected components or detect bipartiteness using only $O(n \log n)$ bits of space (as observed by [Fei+05]).

In a surprising breakthrough, Ahn, Guha, and McGregor [AGM12] showed that these problems can also be solved on turnstile edge streams using only a $\text{polylog}\, n$ factor more space.

Maximum Matching. Perhaps the most studied graph streaming problem is that of finding approximate maximum matchings. This is a fundamental graph problem with many applications, including a connection to matrix rank computation.

On insert-only edge streams, the best known algorithm is a simple greedy strategy - which finds a 2-approximation in $O(n \log n)$ bits of space. A lower bound by Kapralov [Kap13] (improving on [GKK12]) shows that finding a better than $\frac{e}{e-1}$-approximation requires us to use $\Omega(n^{1+1/\log\log n})$ bits of space (even in the vertex incidence list model).

The only known strategies for beating the 2-approximation barrier in the streaming setting either require multiple passes over the data [McG05; KMM12; AG13; KT17], or require us to assume that the input stream arrives in a uniform random order [KMM12; Kon18; Ass+19; Gam+19; Far+20]. Even allowing $n^{2-\epsilon}$ bits of space, it is

not known if we can do better for a single pass streamed in an adversarial order, and closing this gap remains an important open problem.

If we only wish to estimate the maximum matching *size* (but do not need to recover the edge set) Kapralov, Khanna, and Sudan [KKS14] showed how to achieve an $O(\text{polylog}\, n)$-approximation to the size on a *random order* edge stream using only $O(\text{polylog}\, n)$ bits of space.

For the weighted variant of the maximum matching problem (where each inserted edge has an associated weight), a series of results [Fei+05; McG05; Zel12; Eps+11; CS14] provided successively better approximation algorithms for insert-only edge streams, culminating in a breakthrough by Paz and Schwartzman [PS17] (with improved analysis by [GW19]) which showed how to $(2+\epsilon)$-approximate weighted maximum matching using only $O(n \log n \cdot \epsilon^{-1} \log \epsilon^{-1})$ bits of space. This brings the weighted case to parity with the unweighted case, at least for the moment.

Introducing edge deletions makes the problems significantly harder. One of the early graph streaming works by Ahn, Guha, and McGregor [AGM12] proposed a multi-pass turnstile algorithm. The single pass turnstile complexity of c-approximate maximum matching was unresolved until Assadi, Khanna, Li, and Yaroslavtsev [Ass+16] showed $\tilde{\Theta}(n^2/c^3)$ bit upper and lower bounds (with simultaneous weaker upper and lower bounds by [Kon15] and a different optimal algorithm by [Chi+15]).

Chapter 5 of this thesis contains a collection of results for the approximate maximum matching problem.

In Sections 5.2 and 5.3, we consider an edge streaming model that allows a *bounded* number of deletions - something part way between insert-only and turnstile. We show optimal deterministic algorithms for weighted and unweighted matchings in this model. We also provide progress towards a more powerful randomised algorithm in the form of a protocol for a closely related two-party communication model.

In the final Section 5.4, we return to the full turnstile edge streaming model. The lower bound of [Ass+16] has several important caveats, which we overcome by reproving the bound with a completely different strategy. Our result improves upon the existing bound in three ways: (1) it is much simpler - not relying on powerful tools such as *Ruzsa-Szemerédi graphs*, (2) it applies even to streams of bounded length or bounded edge multiplicity - e.g. simple graphs, and (3) we slightly tighten the bound up to exactly $\Omega(n^2/c^3)$ bits.

Independent Sets. Halldórsson, Halldórsson, Losievskaja, and Szegedy [Hal+10] showed how to find a "combinatorially optimal" independent set from an insert-only edge stream in $O(n)$ bits of space, in the sense that the returned set is as large as can be guaranteed to exist from only knowledge of the graph's degree sequence. However,

this size can be as bad as an $O(n)$-approximation to the maximum independent set size in the worst case.

Halldórsson, Sun, and Szegedy [Hal+12] showed that the problem of c-approximating the maximum clique size (or the size of the largest independent set) on an insert-only edge stream has a space complexity of $\tilde{\Theta}(n^2/c^2)$ bits.

Braverman, Liu, Singh, Vinodchandran, and Yang [Bra+18] also consider the problem of c-approximating the maximum clique size. For the vertex incidence list model, they show that $\Omega(m/c^3)$ bits are required - where m is the number of edges. However, this bound only seems to apply when $m = O(n)$.

In another line of work, several authors have considered the related problem of estimating the maximum number of non-overlapping intervals in a stream of intervals. This can be seen as the maximum independent set problem applied to the geometric intersection graph of the intervals. Emek, Halldórsson, and Rosen [EHR16] showed that the best approximation ratio that can be achieved using space proportional to the output size is 2 for arbitrary intervals and $\frac{3}{2}$ for proper intervals. Later Cabello and Pérez-Lantero [CPL17] showed how the solution sizes can be $(2+\epsilon)$ and $(\frac{3}{2}+\epsilon)$-approximated for streams of arbitrary and unit sized intervals (respectively) using only $O(\text{polylog}\, n)$ bits of space. These approximation ratios were also shown to be optimal.

In Chapter 4 of this thesis, we consider several new directions on independent set problems.

Section 4.3 contains new lower bounds for maximal matching in the insert-only edge streaming model (simultaneous with [ACK19]) and for approximate maximum matching in the explicit vertex streaming model.

Section 4.4 contains upper and lower bounds for estimating the maximum independent set size of graphs that have *bounded independence*.

Finally, Section 4.5 contains new upper and lower bounds for square intersection graphs received as a stream of squares and interval intersection graphs received as an explicit vertex stream.

Colouring. We are not aware of much work on graph colouring problems in the streaming setting, in particular there were no claimed bounds on the streaming complexity of estimating the chromatic number of a graph.

One area with a few known results is the problem of finding a $(\Delta+1)$-vertex colouring, where Δ is the maximum degree (such a colouring is always guaranteed to exist). A folklore $O(\log n)$-pass algorithm for computing this kind of colouring in insert-only edge streams using $O(n \log n)$ bits of space was the best known approach until a

remarkable breakthrough by Assadi, Chen, and Khanna [ACK19] showed that one can be computed in a single pass using only $O(n \operatorname{polylog} n)$ bits of space. Simultaneously, [BG18] showed weaker results for the problem of $(1+\epsilon)\Delta$-colouring.

Our main lower bound in Section 4.3.2 also applies to the problem of estimating the chromatic number of a graph in an insert-only edge or explicit vertex stream.

Subgraph Counting. Another very widely studied graph streaming problem is the subgraph counting problem: given a particular subgraph structure, estimate the number of induced subgraphs of the full streamed graph which match this structure. See, for example: [BYKS02; Kan+12; CJ17; BC17].

Parameterisation

Many interesting graph problems unfortunately have no efficient streaming algorithm in general. However, we might still be able to find *parameterised streaming algorithms* for these problems which are efficient if we consider the parameter to be fixed - in analogy to the *fixed-parameter tractable* (FPT) complexity class for offline algorithms. Works in this direction include: [Chi+14; FK14; Chi+16].

This connects directly with our work in Chapter 5, where we consider algorithms for maximum matching parameterised by the number (or weight) of deletions in the stream. We can also draw looser links with parts of Chapter 4, where we consider efficient algorithms for special graph classes.

Lower Bound Techniques

One very useful and interesting fact about the streaming model is its strong connection to the study of communication complexity (see [Yao79]).

Suppose we divide the input stream in half and consider the state of the algorithm at the mid-way point. The algorithm's current state can be seen as a message from the past to the future, from the first half of the stream to the second. If the streaming algorithm works for any possible input stream, then this intermediate state must be at least large enough to solve the corresponding two-party communication problem. A similar argument follows if we want to split the stream into $k > 2$ pieces, except with the k-party communication complexity.

Luckily, tight bounds are known for the communication complexity of many basic problems, allowing us to create strong streaming lower bounds. Commonly used hard problems include: index [KNR95], set disjointness [KS92; AMS99; CKS03], gap hamming [IW03; CR12], augmented index [BY+04a], hidden boolean hypermatching [VY11], and pointer jumping [DJS98; Cha07].

In Section 4.2, we introduce a new k-party communication protocol called *chained index*, which we use to prove several new streaming lower bounds in other parts of the chapter.

For problems on turnstile streams we have an additional powerful tool. Li, Nguyen, and Woodruff [LNW14] explained why almost all known turnstile streaming algorithms are linear sketches. They proved that turnstile streaming algorithms are essentially equivalent to linear sketches: any turnstile streaming algorithm (with some important caveats - though partially lifted by [Ai+16]) can be transformed into a linear sketch which uses only slightly more space (a logarithmic factor).

The upshot of this result is that all lower bounds on linear sketches are also lower bounds on turnstile streaming algorithms, providing a further way to show streaming lower bounds.

In Section 5.4, our main result is a reproof of an existing lower bound for the maximum matching problem. While the original proof made use of the turnstile-sketching equivalence, our new proof does not: instead, we use a more direct proof from a two dimensional variant of augmented index. In particular, this means our new bound avoids the caveats associated with the equivalence result. See [DK20] for more detailed discussion.

1.3 Thesis Roadmap

In this thesis, we take a deeper look at three particular streaming problems: the correlation outliers problem in matrix streams, and the maximum independent set and maximum matching problems in graph streams.

We begin the main body of the thesis with Chapter 2, formalising the models and techniques we will use in rest of the thesis.

Chapter 3: Correlation Outliers. We introduce and study the *correlation outliers* problem. Inspired by a work by Pagh [Pag13], which demonstrated an approach for efficiently finding large entries in a covariance matrix, we considered strategies for the correlation version of the same problem.

Chapter 4: Independent Sets. We consider problems on independent sets in graphs. Since the maximum independent set problem for general graphs is intractable both offline and in the general edge streaming model, we try to attack the problem from other directions. We consider special graph classes and different vertex streaming models where the problem might be more feasible, beginning a comprehensive evaluation of the space complexity of maximum independent set in these various settings.

Chapter 5: Maximum Matching with Bounded Deletions. Finally, we consider the maximum matching problem. Like several other streaming problems, maximum matching exhibits a gap between what is possible in streams with deletions and streams without deletions. A natural question is then: can we describe a trade-off for models where some deletions are allowed, but not arbitrary deletions? In recent work [JW18] showed that the answer to this question is yes for a selection of vector streaming problems (although the problems in question only exhibit a slim $\log n$ factor gap in complexity). This chapter represents our attempts to bring similar closure to the maximum matching problem (where there is a large quadratic gap).

1.4 Recent Developments

Since the original submission of this thesis, there have been a couple of new developments closely connected to the work presented here.

Application of Chained Index. Feldman, Norouzi-Fard, Svensson, and Zenklusen [Fel+20] presented new work on the one-way communication complexity of the submodular maximisation problem. Their multi-party lower bound relies on a reduction to the chained-index problem we proposed in Section 4.2. They include a new much simpler proof of hardness for the chained-index problem. Their proof gives the same complexity bound as Theorem 4.2.3 but removes the need to bound k.

Extension of Techniques from Chapter 5. In a new publication [DK20] we showed a simpler method of proving the maximum matching lower bound from Section 5.4, using an algorithmic reduction instead of an information theoretic argument. We were also able to extend the technique to obtain a tight lower bound for the minimum vertex cover problem.

2 Preliminaries

2.1 Models and Notation

2.1.1 Streaming Algorithms

In its most general form, the streaming model can be formalised as follows:

Definition 2.1.1 (Streams)**.** *A **stream** $\mathcal{S}$ of length m consists of a sequence of m updates to some underlying data structure:*

$$\mathcal{S} = \langle u_1, u_2, \cdots, u_m \rangle$$

The exact nature of the updates can vary significantly depending on the specific model, but is typically some kind of incremental change to a basic element of the data structure in question. The updates are received one-by-one to be processed.

One key characteristic of the streaming model is that the recipient has no control over the order of arrival. The order is always either arbitrary or random.

Definition 2.1.2 (Stream Order)**.** *Consider a stream **S**. It has:*

- ***Adversarial order**, if any particular order of the stream updates is allowed - potentially a worst-case configuration.*
- ***Random order**, if the stream items have been permuted uniformly at random.*

In this thesis, we will exclusively consider adversarial order streams.

The goal is to compute functions of the stream, which is achieved using streaming algorithms. Every streaming algorithm consists of three parts: an initialisation routine, an update routine, and a query routine. The initialisation routine is run before the stream starts, to set up data structures or random seeds. The update routine is called repeatedly to process each stream item received. Finally, at the conclusion of the stream, the query routine is called and the algorithm produces its output.

Definition 2.1.3 (Streaming Algorithms)**.** *A **streaming algorithm** for a problem consists of three parts:*

- *An **initialisation routine**: set up a data structure $\mathcal{D}$.*
- *An **update routine**: make changes to $\mathcal{D}$ to reflect the receipt of a new stream update item u_i.*
- *A **query routine**: use $\mathcal{D}$ to produce the correct output for the problem.*

If we initialise the algorithm and then call the update routine on each stream item in turn, then when we later run the query routine it should then produce the desired problem output.

Our main performance measure is the space cost of the algorithm - the maximum space required to maintain $\mathcal{D}$ for any possible input stream. However, the maximum update time and the maximum query time are also of interest.

While we will primarily consider streaming algorithms that are only allowed to see the stream sequence once, we will sometimes consider the effect on performance of allowing the stream to be viewed multiple times.

Definition 2.1.4 (One-Pass and Multi-Pass Streaming)**.** *A streaming algorithm is said to be:*

- ***one-pass** if the algorithm sees each stream update one-by-one in sequence a single time.*
- *p-**pass**, for integer $p \geq 1$, if the algorithm is allowed to see the entire stream sequence p times consecutively, with the updates given in the exact same order each time.*

Observe that streaming problems can always be solved by a strategy of storing the entire streamed input and then running the best known offline algorithm at query time. So any non-trivial streaming algorithm should use space strictly *sublinear* in the size of the input data.

2.1.2 Vector and Matrix Streams

A vector stream is simply a sequence of incremental changes to the entries of an initially 0 vector, and matrix streams can be defined similarly.

To avoid repeating ourselves, we will define "tensor streams", which generalise these notions to multidimensional arrays. Then vector streams and matrix streams are simply special cases.

Definition 2.1.5 (Tensor Streams)**.** *A length-m* ***tensor stream*** *$\mathcal{S}$ describing an $(n_k)_{k=1}^d$-dimensional tensor X is a sequence of updates, each consisting of an index $I_j \in \prod_{k\in[d]}[n_k]$ and a change $\Delta_j \in \mathbb{R}$:*

$$\mathcal{S} = \big\langle\, (I_1,\, \Delta_1),\, (I_2,\, \Delta_2),\, \cdots,\, (I_m,\, \Delta_m)\,\big\rangle$$

Starting from the initial tensor of all zeros, each update $(I_j,\, \Delta_j)$ tells us to add Δ_j to entry I_j of the tensor. By the end of the stream, this tensor will be equal to X.

For each $i \in \prod_{k\in[d]}[n_k]$ let $U_i = \{j \in [m] \,|\, I_j = i\}$ be the set of positions of updates which made a change to entry i of the tensor. Then each entry i of the described tensor X is given by:

$$x_i = \sum_{j\in U_i} \Delta_j$$

When $d = 1$, we call this a ***vector stream*** *and when $d = 2$, we call it a* ***matrix stream****.*

Depending on the application, we may be interested in various different restrictions on this basic model. For example, we may have a non-negativity constrain on the entries. We list some common variants.

Definition 2.1.6 (Tensor Stream Types)**.** *A vector stream $\mathcal{S}$ is said to be:*

- *A* ***turnstile stream****, when no particular constraints are placed on the allowed updates.*
- *A* ***strict turnstile stream*** *or* ***cash-register stream****, when the final vector X must have all non-negative entries: $X_i \geq 0$.*
- *An* ***insert-only stream****, when all updates must be positive: $\Delta_j > 0$.*
- *An* ***insert-once stream****, when each index occurs exactly once: $|U_i| = 1$*

For matrix streams specifically, we consider two further variants: row streams and column streams. These are a type of insert-once stream where we treat the rows and columns of the matrix (respectively), rather than individual entries, as the basic elements which are updated by the stream. So rather than the matrix arriving entry-by-entry, it arrives either row-by-row or column-by-column.

Definition 2.1.7 (Row and Column Streams)**.** *A **row stream** $\mathcal{S}_R$ describing an $(n \times m)$-dimensional matrix X consists of a sequence of n updates, each consisting of a length m row vector r^j and a row index $\sigma_j \in [n]$:*

$$\mathcal{S}_R = \big\langle (r^1, \sigma_1),\ (r^2, \sigma_2),\ \cdots,\ (r^n, \sigma_n) \big\rangle$$

Each row index occurs exactly once, representing the rows arriving in some permuted order. So the matrix X is simply the block matrix constructed by stacking the rows r^{σ_j} from $j = 1$ to $j = n$.

*Similarly, a **column stream** $\mathcal{S}_C$ describing a X consists of a sequence of m updates, each consisting of a length n column vector c^j and a column index $\sigma_j \in [m]$:*

$$\mathcal{S}_C = \big\langle (c^1, \sigma_1),\ (c^2, \sigma_2),\ \cdots,\ (c^m, \sigma_m) \big\rangle$$

Again, each column index occurs exactly once, representing the columns arriving in some permuted order. So the matrix X is now simply the block matrix constructed by stacking the columns c^{σ_j} from $j = 1$ to $j = m$.

To allow streaming algorithms in this model to still potentially achieve $o(m)$ and $o(n)$ bits of space usage (respectively), we assume each arriving row or column is itself received as a substream of individual entries.

Row and column streams can be thought of as insert-only matrix streams where the order of arrival is partially structured. Rows (or columns) must arrive contiguously, but otherwise the arrival order can still be chosen arbitrarily or randomly as usual.

2.1.3 Graph Streams

First we will set down our graph notation. Notice in particular that we define graphs to be multi-graphs by default, unless we specify them to be simple.

Definition 2.1.8 (Graphs)**.** *A* ***graph*** $G = (V, E)$ *consists of a vertex set* V *and an edge multi-set* E*.*

- *Let the* ***neighbourhood*** *of* $u \in V$ *in graph* G *be given by the set of all vertices with an edge to* u*:*

$$N_G(u) = \{v \in V \mid \{u, v\} \in E\}$$

- *Similarly, define the* ***neighbourhood*** *of vertex set* $U \subset V$ *by the set of all vertices outside of* U *with an edge to at least one vertex in* U*:*

$$N_G[U] = \left(\bigcup_{u \in U} N_G(u) \right) \setminus U$$

- *Let the subgraph* ***induced*** *by* $U \subset V$ *in graph* G *be given by:*

$$G[U] = (U, E_U) \qquad \text{where} \qquad \{\{x, y\} \in E \mid x, y \in U\}$$

Although graphs can be represented as adjacency matrices, it will be useful to consider graph streams separately from vector and matrix streams.

We identify three graph stream models which we will use in this thesis. The first is called the edge streaming model and can be seen as a matrix stream of the adjacency matrix, although we never allow negative edge counts.

Definition 2.1.9 (Edge Stream)**.** *A length-*m ***edge stream*** $\mathcal{S}$ *describing an* n*-vertex graph* $G = (V, E)$ *is a sequence of updates, each consisting of an edge* $e_j = \{u, v\}$ *and a sign* $s_j \in \{-1, 1\}$*:*

$$\mathcal{S} = \big\langle\, (e_1,\, s_1),\, (e_2,\, s_2),\, \cdots,\, (e_m,\, s_m) \,\big\rangle$$

Starting from the empty multiset of edges $Y = \emptyset$*, each update* $(e_j,\, s_j)$ *tells us to either add edge* e_j *to* Y *(if* $s_j = 1$*) or remove* e_j *from* Y *(if* $s_j = -1$*). By the end of the stream, we have that* $Y = E$*.*

We enforce that the stream is only allowed to delete edges currently in Y*.*

Like vector and matrix streams, there are several variants placing different constraints on the the allowed insertions and deletions.

Definition 2.1.10 (Edge Stream Variants)**.** *An edge stream $\mathcal{S}$ is said to be:*

- *A **turnstile stream**, when we place no further restrictions on the stream.*
- *An **insert-only stream**, when no edges are ever deleted: $s_j = 1$.*
- *A **simple stream**, when the intermediate edge multiset Y never contains duplicate edges. In particular, the final graph G must then be a simple graph.*

Unlike vector and matrix streams, edge turnstile streams are already "strict" turnstile streams, since we do not allow a notion of negative edges for graphs.

Similar to how row and column matrix streams replace the notion of entry updates with that of row or column updates, we consider graph streaming models that treat vertices rather than edges as the basic unit of a graph. Our second and third graph stream models are called explicit and implicit vertex streams.

In an explicit vertex stream, we receive a stream of vertices along with any edges incident upon them and previously received vertices. In this way we receive an increasing induced subgraph.

Definition 2.1.11 (Explicit Vertex Stream)**.** *An **explicit vertex stream** $\mathcal{S}$ describing an n-vertex graph $G = (V, E)$ is a sequence of updates, each consisting of a vertex v_j and the set of edges E_j between it and previous vertices:*

$$\mathcal{S} = \langle\, (v_1,\, E_1),\, (v_2,\, E_2),\, \cdots,\, (v_m,\, E_m) \,\rangle$$

Each E_j is given by:
Starting from the empty set of vertices $X = \emptyset$ and the empty set of edges $Y = \emptyset$, each update (v_j, E_j) tells us to add vertex v_j to X and edges E_j to Y. Then, by the end of the stream, $X = V$ and $Y = E$.

As with column and row matrix streams, to allow $o(n)$ space algorithms, we assume that each E_j is received as a substream of edges.

We assume that each of the edge sets are not multi-sets, i.e. they contain no repeat edges. So for us explicit vertex streams are always simple streams. Similarly, while we could define some notion of deletions for explicit vertex streams, we will not consider any such model.

Implicit vertex streams follow a similar structure to explicit vertex streams except, as the name suggests, the graph edges are defined implicitly rather than given directly. Vertices are each associated with an identifier, but we are given an adjacency function which allows us to determine the presence or absence of edges from the identifiers.

Implicit streams are a natural way of modelling graphs which arise from geometric contexts. For example, in a geometric intersection graph each vertex is associated with some geometric object. Then a pair of vertices have an edge exactly when their objects intersect. The stream of geometric objects along with the intersection adjacency function then defines an implicit vertex stream.

Definition 2.1.12 (Implicit Vertex Stream)**.** *An **implicit vertex stream** $\mathcal{S}$ describing an n-vertex graph $G = (V, E)$ is a sequence vertex labels v_j selected from some universe $\mathcal{U}$:*

$$\mathcal{S} = \langle v_1,\ v_2,\ \cdots,\ v_n \rangle$$

We are also equipped with oracle access to some symmetric adjacency function:

$$f : \mathcal{U} \times \mathcal{U} \to \{0, 1\}$$

Then, a particular edge $e = \{v_i, v_j\}$ is included if and only if $f(v_i, v_j) = 1$. So the resulting graph is $G = (V, E_f)$ with vertices $V = [n]$ and edges:

$$E_f = \{\{x, y\} \subset V \mid f(x, y) = 1\}$$

As an example, consider the universe $\mathcal{U} = \mathbb{R}^2$ consisting of points on the real plane. By using the adjacency function

$$f(x, y) = \begin{cases} 1 & \text{if } \|x - y\|_2 \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

we obtain streams representing unit-radius disk intersection graphs. Each stream update $u \in \mathcal{U}$ tells us to add a vertex corresponding to a closed unit disk centered at u, and a pair of vertices have an edge exactly when their respective disks intersect. Notice in particular that the same identifier u could appear multiple times in the stream to indicate multiple different vertices associated with identical disks.

Like the explicit version we only consider implicit vertex streams that are simple and insert-only, although we do note that a variant of implicit vertex streams with deletions would be quite natural and has been studied for the case of interval and disk intersection graphs by Bakshi, Chepurko, and Woodruff [BCW19].

2.1.4 Communication Models

Many tight lower bounds are known on the space complexity of streaming problems, and this is thanks to a strong connection to communication theory. To exploit this connection, we will need to define models and notation for communication problems and protocols.

The main communication model we are interested in for streaming lower bounds is the k-party one-way model. For any given problem, the input will be split among k different parties, who must then perform local computations and communicate with each other to determine the correct problem output. The local computation and storage budget for each party is not limited, but we heavily restrict the communication. The parties must send their messages in a chain, each sending exactly one message directly to their successor, and speaking in order from the head of the chain to the tail. Then the final party in the chain produces the output.

Definition 2.1.13 (One-Way Communication Model)**.** *The* ***k-party one-way communication model*** *supposes that we have k parties called $P^1, P^2, \cdots, P^k$ who each hold a respective input $X^1, X^2, \cdots, X^k$.*

Starting from $i = 1$ up to $i = k - 1$, each party P^i sends a single message to P^{i+1}. Then P^k produces the overall output. Each party may perform unlimited local computation in unlimited local storage, we are only interested in the total number of bits communicated.

A communication protocol is simply an algorithm for each party, describing how to process their received message and local input in order to produce their output message.

Definition 2.1.14 (Communication Protocol)**.** *A* ***k-party one-way communication protocol*** *for a problem is a (possibly random) procedure describing the output message each player should transmit for a given local input and received message. The algorithm solves the problem if, for any input, by having each player follow the procedure we guarantee that the final party will return the desired output.*
The cost of a given communication protocol is the maximum number of bits communicated by all players in total.

Another model variant we will briefly mention is the simultaneous communication model.

Definition 2.1.15 (Simultaneous Communication)**.** *The* ***k-party simultaneous communication model*** *supposes that we have k parties called $P^1, P^2, \cdots, P^k$ who each hold a respective input $X^1, X^2, \cdots, X^k$.*

Every party independently sends a single message to a co-ordinator, who must them determine the correct output to the problem. Again, we are interested in the total number of bits communicated.

The simultaneous model is important in the study of turnstile streams, due to a result of Li, Nguyen, and Woodruff [LNW14]. They showed that any turnstile streaming algorithm (with some caveats) can be simulated by a linear transformation. In particular, this means that any lower bounds for linear transformations are also lower bounds for all turnstile streaming algorithms. Thus, lower bounds shown in the simultaneous communication model apply to the turnstile streaming model, since any linear transformation can be implemented as a simultaneous communication protocol.

The original result did not apply to strict turnstile streams, or streams with bounded values, or multiple passes but more recently Ai, Hu, Li, and Woodruff [Ai+16] extended the reduction to apply to strict turnstile streams, and multiple passes.

2.1.5 Error and Approximation

Our randomised streaming algorithms and communication protocols typically have a small chance of failing. When a particular algorithm or protocol has a δ or smaller chance of not producing the desired output, we say it is δ-error.

Definition 2.1.16 (Error)**.** *We say an algorithm is δ-error if the algorithm produces the correct output with probability at least $(1 - \delta)$. The rest of the time the algorithm may return nothing, or may return an incorrect or invalid output.*

For some algorithms we may wish to further qualify the different error cases - particularly when an algorithm has a large chance of failing, but usually "fails safe". For example, an algorithm for some maximisation problem may be $\frac{1}{3}$-error, but the failure cases may only result in incorrectly claiming an infeasible solution as feasible with a much lower probability of $1/n^2$.

For different maximisation, minimisation, and estimation problems, we want to have a common notion of relative approximation. The usual approach is to say an estimator $\tilde{x}$ of a quantity x is a c-approximation whenever it satisfies $x/c \leq \tilde{x} \leq cx$. However, for maximisation and minimisation problems we often ask for the algorithm to output a feasible solution, meaning that it is impossible for us to over or under-estimate (respectively). This can make it awkward to compare upper bounds for outputting a feasible approximately-maximum solution (which use a one-sided error) with lower bounds for size estimation (which are often expressed with a two-sided error). For this reason, we elect to use the following notion of approximation:

Definition 2.1.17 (Approximation)**.** *We say an algorithm produces a c-approximation to a quantity x if there are constants $c_1, c_2 \geq 1$ with $c_1 \cdot c_2 = c$, such that the algorithm output $\tilde{x}$ satisfies:*

$$x/c_1 \leq \tilde{x} \leq x \cdot c_2$$

whenever it is correct.

So for us, the range of allowed values has a relative-width of c for both one-sided and two-sided approximations. Observe that the particular choice of c_1 and c_2 is not too important (except for when a feasible solution must be output), since any algorithm can have its output rescaled to lie in a different range.

If we wanted to translate any of our upper or lower bounds to the two-sided $[x/c,\ xc]$ variant, simply replace each appearance of c in the space bound with c^2.

We will also sometimes use an additive notion of approximation.

Definition 2.1.18 (Additive Approximation)**.** *We say:*

$$x = y \pm \epsilon \qquad \iff \qquad y - \epsilon \leq x \leq y + \epsilon$$

2.2 Techniques

We will now provide a brief review of the main techniques and results from the fields of streaming algorithms and communication complexity which our work will utilise and build upon.

2.2.1 Tail Bounds

Often, in the evaluation of randomised algorithms, we need to show that our estimators have a low probability of falling outside the desired approximation range. For this, it is helpful to have a toolbox of inequalities for bounding the tails of distributions.

This first bound we will make use of is the well known Markov Inequality.

Fact 2.2.1 (Markov's Inequality)**.** *Given a non-negative random variable X, we have that for any $a > 0$:*

$$\mathbb{P}[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$$

This basic fact follows from observing that any non-negative random variable that exceeds E/a with probability more than a, must have expectation strictly greater than E.

By applying Markov's Inequality to the random variable $(X - \mathbb{E}[X])^2$, we get Chebychev's Inequality:

Fact 2.2.2 (Chebychev's Inequality)**.** *Given any random variable X, we have that for any $a > 0$:*

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathbb{V}[X]}{a^2}$$

The third and final tail bound we will make use of is the Chernoff-Hoeffding Bound, due to Hoeffding [Hoe94], which allows us to places bounds on a sum of independent random variables.

Theorem 2.2.3 (Chernoff-Hoeffding Bound [Hoe94])**.** *Let $X_1, X_2, \cdots, X_n$ be random variables taking values in $\{0, 1\}$ that are either:*

- *Independent and each equal to 1 with probability p.*
- *Each sampled without replacement from a bag of size $N \geq n$, which holds Np copies of 1 and $N(p-1)$ copies of 0.*

In either case, the following two tail bounds hold:

$$\mathbb{P}\left[\sum_{i\in[n]} X_i \geq (p+\epsilon)n\right] \leq \exp\left(-nD(p+\epsilon \parallel p)\right) \leq \exp(-2\epsilon^2 n)$$

$$\mathbb{P}\left[\sum_{i\in[n]} X_i \leq (p-\epsilon)n\right] \leq \exp\left(-nD(p-\epsilon \parallel p)\right) \leq \exp(-2\epsilon^2 n)$$

where $D(a \parallel b) = a \ln \frac{a}{b} + (1-a) \ln \frac{1-a}{1-b}$.

A more careful analysis of $D(a \parallel b)$ gets us a special case of this bound which is very useful for proving relative error guarantees of sampling-based randomised algorithms.

Corollary 2.2.4 (Theorem 1.1 in [DP09])**.** *Let $X = \sum_{i\in[n]} X_i$ be the sum of the random variables from Theorem 2.2.3. Then we have $\mathbb{E}[X] = np$ and:*

$$\mathbb{P}\left[X < \frac{1}{2} \cdot \mathbb{E}[X]\right] \leq \exp\left(\frac{-\mathbb{E}[X]}{8}\right)$$

$$\mathbb{P}\left[X > 2 \cdot \mathbb{E}[X]\right] \leq \exp\left(\frac{-\mathbb{E}[X]}{3}\right)$$

2.2.2 Sketches

Many of the streaming algorithms known for vectors, have a lot of utility as subroutines in the construction of algorithms for complex objects such as graphs. In particular, several "sketches" will be useful for us. A sketch is simply a linear transformation from a high dimensional space into a lower dimensional space, which (usually approximately) preserves some useful properties or information about the original object.

Definition 2.2.5 (Sketch)**.** *A **sketch** is a random linear transformations $\mathcal{S}$ which preserves some useful information about the input in small space.*
Since it is linear, a sketch of some data can be maintained under turnstile stream updates, and even merged.

The first sketch we will make use of is the so-called AMS sketch due to Alon, Gibbons, Matias, and Szegedy [AMS99].

Theorem 2.2.6 (AMS Sketch [AMS99])**.** *For any ϵ, δ, there exists a sketch S of size $O(\epsilon^{-2} \log \delta^{-1})$ bits, and a query routine $\odot$, such that for any vectors x, y we have that:*

$$S(x) \odot S(y) = \langle x, y \rangle \pm \epsilon \|x\| \|y\|$$

with probability at least $(1 - \delta)$, and therefore:

$$S(x) \odot S(x) = (1 \pm \epsilon)\|x\|^2$$

also with probability at least $(1 - \delta)$.

In particular, the query routine involves computing $O(\log \delta^{-1})$ inner products between sub-vectors of length $O(\epsilon^{-2})$ and taking the median.

*Call such a sketch an (ϵ, δ)-**AMS Sketch**.*

The AMS sketch is known to be optimal, since any more efficient sketch would break streaming lower bounds for estimating the second frequency moment (see work by Kane, Nelson, and Woodruff [KNW10b]).

The second sketch we will make use of is an l_0 sampler. Such a sketch allows us to sample uniformly from the support of a vector. This is extremely useful in graph streaming algorithms, as many graph techniques involve sampling edges from subgraphs or neighbourhoods.

Such a data structure was first demonstrated by Frahling, Indyk, and Sohler [FIS08], but we will use the construction of Jowahri, Sağlam, and Tardos [JST11].

Theorem 2.2.7 (l_0 Sampler [JST11])**.** *For any $0 < \delta < 1$, there exists a sketch S of size $O(\log^2 n \log \delta^{-1})$ bits, which supports a query operation.*
For a vector x, when $S(x)$ is queried the sketch succeeds with probability at least $(1 - \delta)$. Conditioned on succeeding, the sketch returns an index $i \in \text{supp}(x)$ chosen uniformly at random from the support of x. When the sketch fails, it usually returns nothing, but has a small $1/n^3$ chance of returning a false entry $j \notin \text{supp}(x)$.

2.2.3 Communication Complexity

Almost all streaming lower bounds take the same form, due to an extremely strong and useful connection to communication complexity.

Fact 2.2.8. *Suppose we have a streaming algorithm $\mathcal{A}$ that uses B bits of space to solve a particular streaming problem.*
Then this streaming algorithm is also a k-party one-way communication protocol for the same problem, where the stream input is spread arbitrarily among the k parties, and this protocol communicates at most kB bits.

Conversely, this means that a lower bounds of M bits of communication for a problem in the k-party one-way communication protocol is also an M/k bit space lower bound for the corresponding streaming problem.

So by determining the randomised communication complexity of problems, we can develop new lower bounds for the streaming model.

The study of randomised communication complexity was introduced by Yao [Yao79] and has since flourished. Many different streaming problems can be related to a small number of core communication problems, by embedding instances of the simple communication problem into the more complex streaming problem in such a way that any solution to the streaming problem must expose the solution to the communication problem.

We briefly list here the key communication problems which we will use in lower bounds for this thesis.

In the index problem, one party knows a secret set, while the other party has a specific element and they would like to know if it is in the set.

Definition 2.2.9 (Index). *In the N-**bit two-party one-way index** communication problem $\textsc{Index}_N$, we have two parties: Alice and Bob.*

- *Alice knows: $X \subset [N]$*
- *Bob knows: $\sigma \in [N]$*

Alice must send a single message to Bob, who must then correctly output 1 if $\sigma \in X$ or 0 otherwise.

The hardness of index is very well known. It describes the basic fact that we cannot compress an arbitrary vector without losing some information.

Theorem 2.2.10 ([KNR95]). *For $\delta < \frac{1}{2}$, any δ-error randomised one-way communication protocol that solves $\textsc{Index}_N$ must communicate at least $\Omega(N)$ bits.*

Our next problem is called set disjointness. Now there may be more than one party, and every party has a set. The sets are either all pairwise disjoint, or they have a unique point all in common.

Index can be seen as a special case of set disjointness, so any problem with a set disjointness lower bound also has an index lower bound. However, the symmetry of set disjointness allows us to have non-trivial multiple pass complexity, and also naturally supports multiple parties - which is necessary to show hardness for some problems.

Definition 2.2.11 (Set Disjointness). *In the N-**bit** k-**party one-way set disjointness** communication problem $\textsc{Disjoint}_N^k$, we have k parties: $P^1, P^2, \cdots, P^k$.*

For $i \in [k]$, each P^i knows $X_i \subset [N]$.

We are guaranteed that either the sets are pairwise disjoint, or they have a unique intersection. That is, either every pair of sets satisfy $X_i \cap X_j = \emptyset$, or there is some $\sigma \in [n]$ such that every pair of sets satisfies $X_i \cap X_j = \{\sigma\}$.

Communication proceeds one-way. Each party goes in order, from P^1 to P^k, sending one message to their immediate successor. Then, P^k must output 0 if the sets are disjoint, or 1 if they have a unique intersection.

Chakrabarti [CKS03] showed tight hardness for the one-way (single pass) variant, and nearly-tight bounds for the multi-way case.

Theorem 2.2.12 (Set Disjointness [CKS03]). *For $\delta < \frac{1}{2}$, any δ-error randomised one-way communication protocol that solves $\textsc{Disjoint}_N^k$ must communicate at least $\Omega\left(\frac{N}{k}\right)$ bits in total. In particular, at least one message must be of size $\Omega\left(\frac{N}{k^2}\right)$.*

If multi-way messages are allowed, at least $\Omega\left(\frac{N}{k \log k}\right)$ bits must be communicated in total.

2.2.4 Information Theory

To prove new lower bounds in communication complexity, our most powerful tools come from information theory. By carefully defining random variables for different parts of a communication problem and studying how they depend on each other, we can relate the probability of succeeding at solving a given problem to the size of the messages sent. In this section, we will quickly review some important basic facts from information theory.

The first measure we will introduce is *entropy* which, roughly speaking, tells us the "information content" (in bits) of a random variable.

Definition 2.2.13 (Entropy). *Suppose we have two random variables X, Y over domains $\mathcal{X}, \mathcal{Y}$, with probability distributions $\mathbb{P}_X, \mathbb{P}_Y$. Use $\mathbb{P}_{X,Y}$ to refer to the joint distribution of X and Y. Then:*

- *The entropy of X is given by*

$$H(X) = \sum_{x \in \mathcal{X}} \mathbb{P}_X(x) \log\left(\frac{1}{\mathbb{P}_X(x)}\right)$$

- *The conditional entropy of X given Y is*

$$\begin{aligned} H(X \,|\, Y) &= \mathbb{E}_Y\left[H(X) \,|\, Y\right] \\ &= \sum_{y \in \mathcal{Y}} \mathbb{P}_Y(y) \sum_{x \in \mathcal{X}} \mathbb{P}_X(x \,|\, Y = y) \log\left(\frac{1}{\mathbb{P}_X(x \,|\, Y = y)}\right) \end{aligned}$$

Another important concept is that of *mutual information*, which can be thought of as the amount of entropy shared by a pair of random variables.

Definition 2.2.14 (Mutual information)**.** *Now suppose we have three random variables X, Y, Z over domains $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$, with probability distributions $\mathbb{P}_X, \mathbb{P}_Y, \mathbb{P}_Z$. Use $\mathbb{P}_{X,Y}$ to refer to the joint distribution of X and Y. Then:*

- *The mutual information between X and Y is given by*

$$\begin{aligned} I(X:Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} \mathbb{P}_{X,Y}(x,y)\log\left(\frac{\mathbb{P}_{X,Y}(x,y)}{\mathbb{P}_X(x)\mathbb{P}_Y(y)}\right) \\ &= H(X)+H(Y)-H(X,Y) \\ &= H(X)-H(X\,|\,Y) = H(Y)-H(Y\,|\,X) \end{aligned}$$

- *The conditional mutual information between X and Y given Z is*

$$\begin{aligned} &I(X:Y\,|\,Z) = \mathbb{E}_Z\left[I(X:Y)\,|\,Z\right] \\ &= \sum_{z\in\mathcal{Z}} \mathbb{P}_Z(z) \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} \mathbb{P}_{X,Y}(x,y\,|\,Z=z)\log\left(\frac{\mathbb{P}_{X,Y|Z}(x,y\,|\,Z=z)}{\mathbb{P}_X(x\,|\,Z=z)\mathbb{P}_Y(y\,|\,Z=z)}\right) \end{aligned}$$

The following three facts are well known.

Fact 2.2.15. *Mutual information is always non-negative $I(X:Y)\geq 0$ and must be 0 when X and Y are independent.*

Fact 2.2.16. *The mutual information cannot be more than the entropy of either variable $I(X,Y)\leq \min\{H(X),H(Y)\}$.*

Fact 2.2.17. *Suppose X and Y are independent. Then the mutual information between the pair (X,Y) and Z is bounded as follows:*

$$I(X,Y:Z)\geq I(X:Z)+I(Y:Z)$$

Proof. The chain rule for mutual information states that:

$$I(X,Y:Z) = I(Y:Z)+I(X:Z\,|\,Y)$$

So we simply need to show that $I(X:Z\,|\,Y)\geq I(X:Z)$ for the result.
The difference $I(X:Z\,|\,Y)-I(X:Z)$ is known as the interaction information $I(X:Y:Z)$, and is invariant under permutation of the variables, therefore $I(X:Z\,|\,Y)-I(X:Z) = I(X:Y\,|\,Z)-I(X:Y)$. We know $I(X:Y)=0$ (from independence) and $I(X:Y\,|\,Z)\geq 0$ (from non-negativity of mutual information), so the inequality must be true. □

3 Correlation Outliers

3.1 Introduction

3.1.1 Background

One of the core problems in data analysis is the identification of correlated pairs of signals within a large data set. By discovering these kinds of structure within the data we can remove unnecessary features, build better predictive models, and identify unexpected or suspicious behaviour.

In this chapter we consider the problem of identifying pairs of signals with unusually large (in magnitude) Pearson product-moment correlation coefficient within a streamed matrix of signal observation data.

Definition 3.1.1 (Correlation Matrix)**.** *Given an n-by-d matrix X let:*

- *The **mean vector** $\mu(X)$ be the length n vector such that:*

$$\mu(X)_i = \frac{1}{d} \sum_{k \in [d]} X_{i,k}$$

- *The **variance vector** $\mathbb{V}[X]$ be the length n vector such that:*

$$\mathbb{V}[X]_i = \frac{1}{d} \sum_{k \in [d]} (X_{i,k} - \mu(X)_i)^2$$

- *The **correlation matrix** $\operatorname{corr}(X)$ be the n-by-n matrix such that:*

$$\operatorname{corr}(X)_{i,j} = \sum_{k \in [d]} \frac{(X_{i,k} - \mu(X)_i) \cdot (X_{j,k} - \mu(X)_j)}{\sqrt{\mathbb{V}[X]_i \mathbb{V}[X]_j}}$$

Throughout this chapter, we will assume that $d \geq 100n$, and that nd is large enough that storing all entries of X is impractical.

We will also be given a threshold parameter τ, such that any off-diagonal entries of $\operatorname{corr}(X)$ outside of the range $(-\tau, \tau)$ are "outliers" which should be reported.

Definition 3.1.2 (Outliers)**.** *Given an n-by-d matrix X and a threshold τ let the* ***outlier set*** $\mathrm{OUT}(X, \tau)$ *be:*

$$\mathrm{OUT}(X, \tau) = \{\{i, j\} \subset [n] \,|\, \tau \leq |\operatorname{corr}(X)_{i,j}|\}$$

Notice that since the index pairs are sets, we have that $i \neq j$. This means that the diagonal entries of $\operatorname{corr}(X)$, which are always 1 and uninteresting, are excluded from $\mathrm{OUT}(X, \tau)$.

It is not really feasible (without storing essentially the whole stream) to distinguish between a correlation of exactly τ and a correlation that is just very slightly less than τ, so we will give ourselves a little breathing room by allowing (but not requiring) the algorithm to report pairs with a correlation of at least $\tau/4$:

Problem 3.1.3 (Correlation Outliers)**.** *Given an n-by-d matrix X provided as a stream along with threshold parameter $0 < \tau < 1$, return a set of index pairs* P *such that:*

$$\mathrm{OUT}(X, \tau) \subset \mathrm{P} \subset \mathrm{OUT}(X, \frac{\tau}{4})$$

with probability at least $1 - \frac{1}{n}$.

In other words, we want to (with high probability) find all pairs with correlation at least τ, without incorrectly reporting any pairs with correlation smaller than $\tau/4$. The choice of the $1/4$ factor is somewhat arbitrary - using any other constant between 0 and 1 would not meaningfully change the complexity of the problem in general.

How difficult the correlations outliers problem is will vary with the choice of threshold τ, but also depends on the combined weight of the non-outlier entries. We quantify this noise as follows:

Definition 3.1.4 (Residue)**.** *Given an n-by-d matrix X and a threshold τ let the* ***residue*** $\mathrm{RES}(X, \tau)$ *be:*

$$\mathrm{RES}(X, \tau) = \sum_{\{i,j\} \notin \mathrm{OUT}(X,\tau)} \operatorname{corr}(X)_{i,j}^2$$

the sum of the squares of the off-diagonal, non-outlier entries.

3.1.2 Results

In Section 3.2, we consider the space required to solve correlation outliers in different streaming models. All bounds are in terms of n, d, and τ (with any polylog factors hidden).

Stream Type	Space Complexity	Theorems
Row, Column, or Turnstile	$\tilde{\mathrm{O}}(n/\tau^2)$	3.2.1
Row	$\Omega(\min\{n/\tau^2, d\})$	3.2.2
Column	$\tilde{\Theta}(\min\{n/\tau^2, n^2\})$	3.2.3

In Section 3.3, we consider the time and space taken by different correlation outliers algorithms at the query processing step.

Search Strategy	Sketch Size	Query Space	Query Time
Naive	$\tilde{\mathrm{O}}(n/\tau^2)$	$\tilde{\mathrm{O}}(n/\tau^2)$	$\tilde{\mathrm{O}}(n^2/\tau^2)$
LSH (Section 3.3.1)	$\tilde{\mathrm{O}}(n/\tau^2)$	$n^{2-\Theta(\tau)}/\tau^2$	$n^{2-\Theta(\tau)}/\tau^2$
Our Approach (Theorem 3.3.6)	$\tilde{\mathrm{O}}(n^{\frac{5}{3}}/\tau^2)$	$\tilde{\mathrm{O}}(n^{\frac{5}{3}}/\tau^2)$	$\tilde{\mathrm{O}}(n^{\frac{5}{3}}/\tau^2)$

With the caveat that the algorithm of Theorem 3.3.6 only works if the residual weight $\mathrm{RES}(X, \tau)$ does not grow faster than $n^{\frac{4}{3}}$, and the number of outliers $|\,\mathrm{OUT}(X, \tau)|$ does not grow faster than $n^{\frac{2}{3}}$.

3.2 Space Complexity

In this section we demonstrate how a simple direct application of vector sketching techniques solves the correlation outliers problem for turnstile matrix streams. We then show that this strategy is essentially optimal in terms of space costs for all three of the matrix streaming models we consider.

3.2.1 Turnstile Row-Sketching Strategy

Since AMS sketches can be shifted and rescaled, if we store an AMS sketch of each row, we can shift and normalise them and then make inner product queries to estimate the correlations. By trying all n^2 possibilities, we can then find all the outliers.

Theorem 3.2.1. *There is a $1/n$-error randomised turnstile matrix streaming algorithm which finds every index pair in $\mathrm{OUT}(X, \tau)$ using $O(n \log n/\tau^2)$ bits of space, and does not falsely report any entries unless they have magnitude at least $\tau/4$.*

Proof. Randomly select a $(\tau/10, 1/2n^3)$-AMS sketch function S and another $(1/10, 1/2n^2)$-AMS sketch function S'.

Streaming Phase Maintain sketches $S(r^i)$ and $S'(r^i)$ of each row r^i of X. Also store the sum of columns $d \cdot \mu(X)$. This requires $O(n/\tau^2 \cdot \log n)$ bits of space in total.

Query Phase At the conclusion of the stream, we can determine the mean column $\mu(X)$ and use it to retroactively shift all the sketched rows $S(r^i)$ and $S'(r^i)$ to have mean 0.

For row i compute $S(1_d)$ the sketch of the all-ones vector and compute $S(p^i) = S(r^i) - \mu(X)_i \cdot S(1_d)$. By linearity, this gives us sketches of the centered rows. Similarly for $S'(p^i)$.

Next, we use the norm estimation property of AMS sketches to approximate each row variance $\mathbb{V}\left[r^i\right]$ from $S'(p^i)$. We approximate all the variances to within $(1 \pm \frac{1}{2})$ simultaneously with probability at least $(1 - 1/2n)$, by a union bound over the n attempts.

Now we rescale all of the sketches $S(p^i)$ according to the estimated variances to be sketches of approximately normalised and centered rows $S(v^i) \approx S(p^i)/\mathbb{V}\left[r^i\right]$. Each has norm between $\frac{10}{11}$ and $\frac{10}{9}$. Now we perform all pairs of inner product queries between sketches $S(v^i)$. These all meet their AMS guarantee with probability at least $(1 - 1/2n)$, by a union bound over the n^2 attempts.

Any correlation of magnitude at least τ results in an inner product query of more than $\tau/2$, while any correlation of magnitude at most $\tau/4$ results in an inner product query of less than $\tau/2$. Therefore, we can distinguish the cases as promised.

The query takes a total of $O(n^2 \log n/\tau^2)$ time to execute. □

3.2.2 Row Streaming Lower Bound

On the lower bound side of things, we will first consider the row streaming model, where the algorithm receives rows of the observation matrix one-by-one.

The hardness of sketching normalised inner products is normally limited by the amount of precision desired, requiring $\tilde{\Omega}(\tau^{-2})$ bits of space to achieve a τ additive error. Roughly speaking, this holds because you can pack in at most τ^{-2} entries of magnitude τ into a unit weight vector. For our problem, we have n different row vectors to work with, so we can get up to $\Omega(n/\tau^2)$ hardness.

Theorem 3.2.2. *Any $\frac{1}{3}$-error randomised row streaming algorithm which solves the correlation outliers problem must use at least $\Omega(\min\{n/\tau^2, d\})$ bits of space.*

This holds even when there is at most one outlier $|\operatorname{OUT}(X, \tau)| \leq 1$ and all other off-diagonal entries of $\operatorname{corr}(X)$ are 0.

Proof. Suppose there is a $\frac{1}{3}$-error randomised row streaming algorithm $\mathcal{A}$ which returns an entry of $\operatorname{OUT}(X, \tau)$, using $B(n, \tau)$ bits of space for an n-row matrix M.

Without loss of generality, we can choose integer $n > 1$ and real threshold $\tau > 0$, such that $L = 1/\tau^2$ is an integer. Now consider an instance (Y, σ) of the two-party

communication problem $\textsc{Index}_N$ with number of bits $N = (n-1)L$.

Alice's Rows Given Y we have Alice construct $n-1$ row vectors $r^1, r^2, \cdots, r^{n-1}$ - each of length $2N$. Notice this means that we need $2N \leq d$, hence why the lower bound is capped by d.
Each vector will be responsible for encoding L bits of Y, and will be constructed to have mean 0, variance 1, and no correlation with the other rows.
Partition Y into $\bigcup_{i\in[n-1]} Y_i$ where each $Y_i = \{k \in Y \mid (i-1)L < k \leq iL\}$ consists of up to L items from Y.
For each $i \in [n-1]$ and $j \in [N]$ let the j^{th} entry of the i^{th} row be given by:

$$r^i_j = \begin{cases} (2|Y_i|)^{-\frac{1}{2}} & \text{if } j \in X_i \\ -(2|Y_i|)^{-\frac{1}{2}} & \text{if } j - N \in X_i \\ 0 & \text{otherwise} \end{cases}$$

So for each $k \in S_i$, the vector r^i gets an an entry of $(2|Y_i|)^{-1/2}$ at position k and an entry of $-(2|Y_i|)^{-1/2}$ at position $k+N$. All other entries are left 0. This means that every row has mean $\mu(r^i) = 0$ and variance $\mathbb{V}\left[r^i\right] = 1$. Since the sets Y_i are pairwise disjoint, we also have that the vectors are pairwise uncorrelated, as we wanted.
Alice now simulates algorithm $\mathcal{A}$ on some arbitrary stream ordering of the generated rows and then sends the algorithm state $\mathcal{M}$ to Bob. We know this message has bounded size $|\mathcal{M}| \leq B(n, \tau)$.

Bob's Row Given the algorithm state $\mathcal{M}$ and the index $\sigma \in [N]$, Bob will continue simulating algorithm $\mathcal{A}$ to insert one more length N row r^*. We want this row to correlate with another row if and only if $\sigma \in Y$, so let the j^{th} entry of r^* be given by:

$$r^*_j = \begin{cases} 1/\sqrt{2} & \text{if } j = \sigma \\ -1/\sqrt{2} & \text{if } j = \sigma + N \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the row also has $\mu(r^*) = 0$ and $\mathbb{V}[r^*] = 1$, and the only other entries in columns σ and $\sigma + N$ could be from row r^I for $I = \lceil \sigma/L \rceil$, and only when $\sigma \in Y_I$. So the correlation between r^* and each r^i for $i \neq I$ is 0, while the correlation between r^* and r^I is $1/\sqrt{|Y_I|} \geq 1/\sqrt{L} = \tau$ when $\sigma \in Y$ or 0 otherwise.

Output Let X be the combined matrix of the n inserted rows. Recall that the first $n-1$ rows were pairwise uncorrelated, and the final row is correlated (with correlation at least τ) with row $I = \lceil \sigma/L \rceil$ if and only if $Y_\sigma = 1$, but also otherwise uncorrelated.

Therefore, if $Y_\sigma = 1$, then with probability at least $\frac{2}{3}$ algorithm $\mathcal{A}$ return $\{n, I\}$ and, if $Y_\sigma = 0$, then with probability at least $\frac{2}{3}$ algorithm $\mathcal{A}$ returns nothing.
By Theorem 2.2.10, any such protocol must have communicated a message of at least $\Omega(N) = \Omega(n/\tau^2)$ bits, so we have that $B(n, \tau) \geq \Omega(n/\tau^2)$. □

3.2.3 Column Streaming Lower Bound

Now we consider the column streaming model, where the algorithm receives columns of the observation matrix one-by-one in an arbitrary order. We will show that the correlation outliers problem is similarly hard in this model as in the row streaming model, although for slightly different reasons.

We will still hide $O(1/\tau^2)$ bits of information within each of the n data rows. However, since row insertions are not allowed, we also have to hard code $O(1/\tau^2)$ additional point-query rows. Each point-query row is constructed to correlate with one of the bits from every data row, giving us a way to try to identify the state of any given bit.

Since all the bits must be lined up in order for the useful correlations to exists, we also cause many noisy cross-correlations between the data rows. Fortunately, since we can now perform column-insertions, we can choose to retroactively blow up the variance of any row. By increasing the variance of every row except a particular data and query row, we completely drown out all possible correlations except for the potential correlation associated with a particular bit.

Theorem 3.2.3. *Any $\frac{1}{3}$-error randomised column streaming algorithm which solves correlation outliers must use at least $\Omega(n \cdot \min\{\tau^{-2}, n\})$ bits of space.*

This holds even when there is at most one outlier $|\operatorname{OUT}(X, \tau)| \leq 1$ and all other off-diagonal entries of $\operatorname{corr}(X)$ have magnitude at most τ/n^3.

In combination with Theorem 3.2.1, this gives us a tight (up to polylog *factors) bound of $\tilde{\Theta}(\min\{n/\tau^2, n^2\})$ on the space complexity of correlation outliers in the column streaming setting.*

Proof. Suppose there is a $\frac{1}{3}$-error randomised column streaming algorithm $\mathcal{A}$ which returns an entry of $\operatorname{OUT}(X, \tau)$, using $B(n, \tau)$ bits of space for an n-row matrix X. Without loss of generality, we can choose an integer $n > 1$ and a real threshold τ such that $L = 1/\tau^2 < n/2$ is an integer. Now consider an instance (Y, σ) of the two-party communication problem $\textsc{Index}_N$ with number of bits $N = (n - L)L$.

Alice's Columns Given Y we have Alice construct L columns $c^1, c^2, \cdots, c^L$ - each of height n. The first $J = n - L$ entries of each column will be used to encode bits of Y, and the final L entries will be hard coded.
Again, partition Y into $\bigcup_{i \in [J]} Y_i$ where each $Y_i = \{k \in X \,|\, (i-1)L < k \leq iL\}$

consists of up to L items from Y.
For each $i \in [L]$ and $j \in [n]$ let the j^{th} entry of the i^{th} column be given by:

$$c_j^i = \begin{cases} (2|Y_j|)^{-\frac{1}{2}} & \text{if } j \leq J \text{ and } ((j-1)L+i) \in X_j \\ 1/\sqrt{2} & \text{if } j = J+i \\ 0 & \text{otherwise} \end{cases}$$

Alice then simulates algorithm $\mathcal{A}$ inserting each of the generated columns c^i for $i \in [L]$. Alice follows this by also inserting a negated version $-c^i$ of the generated columns for $i \in [L]$. Alice then passes the algorithm state $\mathcal{M}$ to Bob. This message has bounded size $|\mathcal{M}| \leq B(n, \tau)$.

Intermediate State We should pause here to evaluate the current state of the inserted observation matrix X. Label the first J rows as $r^1, r^2, \cdots, r^J$ and the final L rows as $q^1, q^2, \cdots, q^L$. Each r^j for $j \in [J]$ consists of $|Y_j|$ entries of value $(2|Y_j|)^{-1/2}$ and $|Y_j|$ entries of value $-(2|Y_j|)^{-1/2}$ and the rest of the entries 0, giving a mean $\mu(r^j) = 0$ and variance $\mathbb{V}\left[r^j\right] = 1$. Each q^i for $i \in [L]$ consists of one entry of $1/\sqrt{2}$ and one of $-1/\sqrt{2}$ and the rest of the entries 0, also giving a mean of $\mu(q^i) = 0$ and variance $\mathbb{V}\left[q^i\right] = 1$.
A given pair q^i and r^j have correlation $1/\sqrt{|Y_j|} \geq \tau$ if $((j-1)L+i) \in Y_j$ and 0 otherwise. All the rows q^i for $i \in [L]$ are pairwise uncorrelated. Unfortunately, the rows r^j for $j \in [J]$ are potentially all highly correlated, flooding the set $\text{OUT}(X, \tau)$ and drowning out the useful correlation between each q^i and r^j.

Bob's Columns Given algorithm state $\mathcal{M}$ and the index $\sigma \in [n]$, Bob will continue simulating algorithm $\mathcal{A}$ to insert $2n$ more columns. Let $x, y > 0$ be the unique integers such that $x \leq L$ and $(y-1)L + x = \sigma$. Bob wishes to input columns which will cause all correlations other than that between rows q^x and r^y to vanish. To achieve this, he will introduce entries to blow-up the variance of all the other rows.
For each $i \in [n] \setminus \{x+J, y\}$ Bob inserts two columns, one with a single non-zero entry of $n^{7/2}/\tau$ at position i, and one with a single non-zero entry of $-n^{7/2}/\tau$ at the same position. Since the additions to each row sum to 0, the mean will be unchanged. And since each added column has only a single entry, this also does not change any of the inner products between rows.

Algorithm Output Now, consider again the rows of the extended observation matrix X using the same labels as before. As we just stated, every row still has mean 0, and every pair of rows has the same inner product as before (which was at most 1, since their variances were previously all 1). The rows r^y and q^x still have variance $\mathbb{V}[q^x] = \mathbb{V}[r^y] = 1$. However, the rows q^i and r^j for $i \in [L] \setminus \{x\}$ and $j \in [J] \setminus \{y\}$ each have variance $\mathbb{V}\left[r^j\right], \mathbb{V}\left[q^i\right] \geq n^6/\tau^2$.

This means that, while the correlation between q^x and r^y is unchanged, the correlations between any other pair of rows (even when one of the pair of rows is q^x or r^y) is at most τ/n^3. In particular, when $Y_\sigma = 1$ we have that $\text{OUT}(X, \tau) = \{\{x + J, y\}\}$, and when $Y_\sigma = 0$ we have that $\text{OUT}(X, \tau) = \emptyset$.
Since algorithm $\mathcal{A}$ distinguishes these cases with probability at least $\frac{2}{3}$, by Theorem 2.2.10, our constructed protocol must have communicated a message of at least $\Omega(N) = \Omega(n/\tau^2)$ bits, and so $B(n, \tau) \geq \Omega(n/\tau^2)$.
We inserted a total of $2L + 2n \leq 3n$ columns, which is fewer than the allowed $d \geq 100n$. However, we had to assume that $\tau^{-2} < n/2$ at the start, so n is a cap on the τ^{-2} factor. This gives us the overall lower bound.

Tightness To observe that this lower bound is tight, we consider two upper bounds: the $\tilde{\text{O}}(n/\tau^2)$ upper bound of Theorem 3.2.1, and the following trivial $\tilde{\text{O}}(n^2)$ space algorithm.
For each row track the sum of entries and the sum of squares of entries in $\tilde{\text{O}}(n)$ space - this allows us to calculate exactly the mean and variance of each row. As each column x arrives, compute the outer product xx^{T}. Track the sum of all these xx^{T}'s in $\tilde{\text{O}}(n^2)$ space - at the end of the stream, this will be the matrix of all row inner products. From the inner products, the means, and the variances, we can calculate all the correlation entries exactly. □

3.3 Query Time

While the row sketching strategy is space-optimal for our problem, the naive approach of computing all the inner product queries and comparing each against the threshold is very slow, taking $\tilde{\Theta}(n^2/\tau^2)$ time. While this is essentially optimal for the general case where there may be as many as $\Theta(n^2)$ outlier pairs to report, we can hope to do better when the set of near-outliers $\text{OUT}(X, \tau/4)$ is relatively small.

In this section we look at existing techniques for performing fast all-pairs searches, and propose a sketch-based approach that can perform better for certain parameter regimes.

3.3.1 Fast All-Pairs Search

In this subsection, we will discuss how *locality sensitive hashing* (LSH) can be used to achieve a faster query time - though at the cost of using more space while running the query procedure.

Locality Sensitive Hashing The LSH framework was introduced by Indyk and Motwani [IM98] as a way of efficiently solving the *near neighbours problem*. It uses a

special kind of hash function which is biased towards similar vectors colliding more often than dissimilar vectors.

Concretely: pairs of vectors within distance r of each other are considered *near* and collide in the hash function with probability at least p_1, while pairs of vectors more than distance cr apart are *far* and only suffer a hash collision with probability at most p_2. We measure the quality of an LSH function by its *sensitivity* $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.

Now, suppose we have a collection of n length-l vectors. The algorithm of [IM98, Theorem 5] builds us a hash table of size $\tilde{\mathrm{O}}(l \cdot n^{1+\rho})$ bits, taking the same order time to complete. This hash table has two important properties. For each vector x in the collection:

- Each y that is near to x (i.e. $\|x-y\| \leq r$) has a constant probability of colliding with x in the table.
- With constant probability, at most n^ρ different vectors y' that are far from x (i.e. $\|x - y'\| > cr$) collide with x in the table.

Armed with these facts, we can see that by building the hash table and inspecting up to $O(n^\rho)$ collisions for each vector in the collection (taking a total of $\tilde{\mathrm{O}}(l \cdot n^{1+\rho})$ time), we will find a constant fraction of all the near pairs x, y such that $\|x - y\| \leq r$, as long as there are not too many *not-far* pairs with $\|x - y\| \leq cr$. Specifically, we need that each vector in the collection is not-far from at most $O(n^\rho)$ other vectors.

When this holds, we find a constant fraction of the near pairs each time we build and search through a new independent hash table. Therefore, performing this process $O(\log n)$ times will find us all the near pairs with high probability (and we can filter out all false reports by simply checking the actual distance between each candidate pair - for no meaningful additional time cost).

For Euclidean space, the optimal *data-independent* LSH function has a sensitivity of $\rho(r, c) = 1/c^2 + o(1)$ [IM98]. If we allow the hashing function to be *data-dependent* (meaning the hash of each vector can depend on the full data collection), Andoni and Razenshteyn [AR15] showed how to improve this to an optimal sensitivity of $\rho(r, c) = 1/(2c^2 - 1) + o(1)$.

Application to Correlation Outliers So how does this relate to the correlation outliers problem? The key observation is that we can use linear sketches to reduce our correlation outliers problem into an all-pairs near neighbours search problem in Euclidean space.

Our first idea is to reuse the normalisation strategy from Theorem 3.2.1. If we use sketches to track the means and approximate the Euclidean norms of each row, then we can simulate the original matrix X having all rows of mean 0 and Euclidean norm

1 ± 0.01. This only introduces a roughly 2% distortion to each entry in the correlation matrix (with high probability). For the rest of this section, we will assume X always had rows normalised in this way.

Now consider these normalised rows of X. A pair of rows have high (positive) correlation exactly when they have small Euclidean distance between them. Specifically, the cosine rule tells us that rows r_1 and r_2 satisfy:

$$\|r_1 - r_2\|_2 = \sqrt{2 - 2 \cdot \text{corr}(r_1, r_2)}$$

where $\text{corr}(u_1, u_2)$ is the correlation between the rows (which is simply the dot product in this case).

So we can hope to find correlation outliers by searching for pairs of rows which are close in Euclidean distance. We can similarly detect high negative correlation by simply checking for rows which are close to another negated row.

Before we can go any further, we also need to deal with the fact that the rows of X are too big to store. We can use a fast linear Johnson-Lindenstrauss transform [AC06] to compress the input rows of X from d dimensions down to $O(\tau^{-2} \cdot \log n)$, at the cost of distorting the pairwise Euclidean distances by a factor of at most $1 \pm \frac{\tau}{100}$ (with high probability). This transformation is linear, so it does not interfere with the normalisation process outlined above and can be maintained over a turnstile stream.

So now finding the correlation outliers becomes exactly a problem of finding the near neighbours in Euclidean distance, for a collection of n vectors each of length $l = O(\tau^{-2} \cdot \log n)$.

Recalling the definitions of near and far vectors, we need to set $r \geq \sqrt{2 - 2\tau}$, to ensure all correlation outliers count as near neighbours. On the other hand, it's less clear what the right choice for cr is. We clearly need $r < cr < \sqrt{2}$, but the exact choice depends on the distribution of the non-outliers. We need cr to be small enough to exclude all but $O(n^\rho)$ non-outliers.

Whatever the optimal choice of cr is, we end up with:

$$\frac{1}{c^2} = \frac{r^2}{(cr)^2} \geq \frac{2 - 2\tau}{2} = 1 - \tau$$

giving us $\rho \geq 2 - \Theta(\tau)$, even with the improved data-dependent hashing.

This gives an overall space and time cost of $n^{2-\Theta(\tau)}/\tau^2$ to implement this LSH-based fast query strategy. This can be quite efficient for τ close to 1, but is not very useful for detecting weak outliers - even at the extreme where all the non-outliers are 0.

Other Related Work For the special case of Boolean vectors, there are strategies for achieving even faster queries [Val12; KKK18], but we are interested in arbitrary

vectors of reals here. Finding a way to apply these improved techniques to our problem is an interesting direction for future work.

3.3.2 Sketching the Sketches

We will describe an alternative strategy for the case of small threshold τ and vanishing residue $\mathrm{RES}(X, \tau)$. Our idea is to use the stored row sketches of the observation matrix M to build a sketch of the correlation matrix $\mathrm{corr}(M)$ by applying an additional layer of sketching in the query step. Unfortunately, nesting sketches in this way requires us to have polynomially small error in the initial sketches, resulting in a corresponding increase in the required sketch size. This produces a trade-off between query time and sketch size.

For our strategy we need to be able to identify the unique heavy element hiding in a sum of elements, a version of the so-called "heavy-hitters" problem. For this we borrow the scheme used by Pagh [Pag13, Section 4] (with similar ideas used previously in the sparse recovery literature by Gilbert, Li, Porat, and Strauss [Gil+12]). This scheme calls for the use of good error correcting code for encoding and decoding row indices. Since there are at most n rows, we would like codes that take inputs of length $B = \lceil \log n \rceil$ bits, produce codewords of length $O(B)$, and can successfully decode a codeword which has had up to some constant fraction of its bits flipped. Such codes exist, for example the expander codes of Sipser and Spielman [SS96].

Fact 3.3.1 (Error-Correcting Code)**.** *There exist constants $\lambda, W > 0$ such that for every integer B there are a pair of functions - an encoding function $\mathcal{E}$ and a decoding function $\mathcal{D}$:*

$$\mathcal{E} : [2^B] \to \{0,1\}^{WB}$$
$$\mathcal{D} : \{0,1\}^{WB} \to [2^B]$$

such that for any input $x \in [2^B]$ and error sequence $y \in \{0,1\}^{WB}$ satisfying $|\operatorname{supp}(y)| \leq \lambda WB$, we have that:

$$\mathcal{D}(\mathcal{E}(x) \oplus y) = x$$

Where $\oplus$ is the entry-wise XOR operation, which can be thought of as flipping the bit of $\mathcal{E}(x)$ whenever the corresponding bit of y is 1.

Since there are only $2^B < 2n$ possible inputs, we can simply build a pair of $O(n \log n)$ size lookup table of codewords to allow fast encoding and decoding. The encoding table is a list of codewords ordered by input, and requires constant time to look up and encoding. The decoding table is a list of inputs ordered by codeword, and the decoding can be found by a fast binary search.

Definition 3.3.2. *For any integer $n > 0$, use* $\textsc{Encode}_n : [n] \to \{0,1\}^{W\lceil \log n \rceil}$ *and* $\textsc{Decode}_n : \{0,1\}^{W\lceil \log n \rceil} \to [n]$ *be a fixed choice of encoding and decoding functions with properties guaranteed by Fact 3.3.1 and which require $O(n \log n)$ space and $O(\log^2 n)$ time to compute.*

Use $\textsc{Encode}_n(i; b)$ *to refer to the b^{th} bit of* $\textsc{Encode}_n(i)$.

We define the notion of a "grid sketch" which compresses a matrix down, but still allows us to identify outlier entries quickly. This can be seen as performing a count sketch on both the rows and columns of the matrix.

Definition 3.3.3 (Grid Sketch)**.** *Suppose we have number g of groups which divides n, and a square n-by-n matrix X.*
Choose any pair of partitions of $[n]$ into g groups of size n/g and let $\Gamma_x, \Gamma_y : [g] \to 2^{[n]}$ be the maps of group numbers to group members. Also choose a pair of sign functions $\sigma_x, \sigma_y : [n] \to \{-1, +1\}$.

Then for sketch maps $\mathcal{B} = (\Gamma_x, \Gamma_y, \sigma_x, \sigma_y)$ and for each $u, v \in [g]$ and $b \in [L]$ let $\textsc{GridSketch}_g^n(X; \mathcal{B}; u, v, b)$ *be given by:*

$$\sum_{i \in \Gamma_x(u)} \left(\sum_{j \in \Gamma_y(v)} (\sigma_x(i) \cdot \sigma_y(j) \cdot X_{i,j} \cdot \textsc{Encode}_n(i; b)) \right)$$

and let $\textsc{GridSketch}_g^n(X; \mathcal{B}; u, v, b + L)$ *be given by:*

$$\sum_{i \in \Gamma_x(u)} \left(\sum_{j \in \Gamma_y(v)} (\sigma_x(i) \cdot \sigma_y(j) \cdot X_{i,j} \cdot \textsc{Encode}_n(j; b)) \right)$$

where $L = 2W\lceil \log n \rceil$ using W from Fact 3.3.1.

Use $\textsc{GridSketch}_g^n(X; \mathcal{B})$ *to refer to the size $(g \times g \times 2L)$ array of all grid sketch values for $u, v, b \in [g]^2 \times [2L]$.*

If we could approximate each grid sketch entry for our correlation matrix with randomly chosen group and sign functions, then we could identify the correlation outliers, as shown by the following lemma.

Algorithm 1: Grid Sketch Decoder

Input: Size $(g \times g \times 2L)$ array Y where $L = W\lceil \log n \rceil$ and threshold τ

$S \leftarrow \emptyset$
for $u, v \in [g]^2$ **do**
  $x \leftarrow$ length L vector of 0's
  $y \leftarrow$ length L vector of 0's
  for $b \in [L]$ **do**
    if $|Y_{u,v,b}| \geq \tau/2$ **then**
      $x_b = 1$
    if $|Y_{u,v,b+L}| \geq \tau/2$ **then**
      $y_b = 1$
  $S \leftarrow S \cup \{(\textsc{Decode}_n(x), \textsc{Decode}_n(y))\}$
return S

Lemma 3.3.4. *Suppose we have a square n-by-n matrix M, a set $I \subset [n]^2$ of size $|I| = k \leq n/4$, and thresholds R, τ such that:*

- *Each diagonal entry $M_{i,i} = 1$*
- *For each $(i,j) \in I$ we have $|M_{i,j}| \geq \tau$*
- *The sum of squares of other entries satisfies*

$$\sum_{(i,j) \notin I \wedge i \neq j} M_{i,j}^2 \leq R$$

Recall L and λ, the constants from Fact 3.3.1. Suppose we can choose some integer g satisfying:

$$\max\left\{16\frac{R^{\frac{1}{2}}}{\tau\lambda^{\frac{1}{2}}},\ 4k\right\} \leq g \leq n$$

Sample independently and uniformly at random some sketch maps $\mathcal{B} = (\Gamma_x, \Gamma_y, \sigma_x, \sigma_y)$ compatible with Definition 3.3.3. Now suppose that for each $u, v \in [g]$ and $b \in [2L]$ we produce an estimate $W_{u,v,b}$ which satisfies:

$$W_{u,v,b} = \textsc{GridSketch}_g^n(M; \mathcal{B}; u, v, b) \pm \frac{\tau}{4}$$

with probability at least $(1 - \lambda/8)$, not necessarily independently.

Let $U = \textsc{GridSketch}_g^n(M; \mathcal{B})$. Now, if we run Algorithm 1 on $Y = W - U$, the output S is a set of size at most g^2 which contains any particular $(i,j) \in I$ with probability at least $\frac{1}{4}$.

Proof. Consider some particular $(i,j) \in I$. Let u, v be the group numbers such that $i \in \Gamma_x(u)$ and $j \in \Gamma_x(v)$. We will show that with probability at least $\frac{1}{4}$ we will have that $(i,j) \in S$.

Outlier Collisions First we consider the chance that another pair $(i', j') \in I$ is also mapped to groups u and v by Γ_x and Γ_y respectively. The happens with probability at most $1/g$. Notice we cannot bound with $1/g^2$, because it could be that $i = i'$ or $j = j'$ (although not both).
Use NoCollide to refer to the event where this does not happen for any other $(i', j') \in I$. By the union bound, we have that:

$$\mathbb{P}\left[\textsc{NoCollide}\right] \geq \left(1 - \frac{k-1}{g}\right) \geq \frac{3}{4}$$

Grid Sketch Entry Noise Now, conditioned on NoCollide, we consider a particular entry $Y_{u,v,b}$ for $b \in [2L]$. Letting $Z_b = \textsc{GridSketch}_g^n(M; \mathcal{B}; u, v, b)$ and

$$z_b^* = \begin{cases} \sigma_x(i) \cdot \sigma_y(j) \cdot M_{i,j} \cdot \textsc{Encode}_n(i; b) & \text{if } b \in [L] \\ \sigma_x(i) \cdot \sigma_y(j) \cdot M_{i,j} \cdot \textsc{Encode}_n(j; b - L) & \text{otherwise} \end{cases}$$

Let $\textsc{GoodEntry}_b$ refer to the event when:

$$|Y_{u,v,b} - z_b^*| \leq |W_{u,v,b} - Z_b| + |Z_b - U_{u,v,b} - z_b^*| \leq \frac{\tau}{2}$$

The first inequality is always true by the triangle inequality.
We will argue that for every $b \in [2L]$ we have:

$$\mathbb{P}\left[\textsc{GoodEntry}_b\right] \geq (1 - \lambda/4)$$

From the estimate claim in the lemma statement, we know that $|W_{u,v,b} - Z_b| \leq \tau/4$ with probability at least $(1 - \lambda/8)$, so it would be enough to show that $|Z_b - U_{u,v,b} - z_b^*| \leq \tau/4$ also with probability at least $(1 - \lambda/8)$.
Observe that since the map from matrices to grid sketch entries is linear, $Z_b - U_{u,v,b} - z_b^*$ is simply a grid sketch entry for M with the diagonals and entry (i, j) all set to 0.
Concretely, if $b \leq L$:

$$Z_b - U_{u,v,b} - z_b^* = \sum_{s \in \Gamma_x(u) \setminus \{i\}} \left(\sum_{t \in \Gamma_y(v) \setminus \{j,s\}} (\sigma_x(s) \cdot \sigma_y(t) \cdot M_{s,t} \cdot \textsc{Encode}_n(s; b)) \right)$$

or if $L < b \leq 2L$:

$$Z_b - U_{u,v,b} - z_b^* = \sum_{s \in \Gamma_x(u) \setminus \{i\}} \left(\sum_{t \in \Gamma_y(v) \setminus \{j,s\}} (\sigma_x(s) \cdot \sigma_y(t) \cdot M_{s,t} \cdot \textsc{Encode}_n(t; b - L)) \right)$$

Each pair of these terms contributing to $Z_b - U_{u,v,b} - z_b^*$ is uncorrelated since each is multiplied by some random sign independently of all other terms (either σ_y for terms in the same column, or σ_x for terms in the same row). So the total variance

is the sum of the entry variances, which are each bounded by the sum of squares of the possible entries (which is at most R) divided by the number of possible values (which is $n^2 - k + 1 \geq n^2/2$).

$$\mathbb{V}\left[Z_b - U_{u,v,b} - z_b^*\right] \leq \left(\frac{n}{g}\right)^2 \cdot \frac{2R}{n^2} = \frac{2R}{g^2} \leq \frac{\tau^2 \lambda}{128}$$

Therefore, by Chebychev's inequality, we have that $|Z_b - U_{u,v,b} - z_b^*| > \tau/4$ with probability at most $\lambda/8$, as we wanted.

Decoding Now, still conditioned on NoCollide, we consider what is the chance that the algorithm adds (i, j) to S in the outer loop corresponding to our particular u, v of interest.

Observe that when $\textsc{GoodEntry}_b$ occurs for $b \in [L]$, we have that $|Y_{u,v,b}| \geq \tau/2$ if and only if $\textsc{Encode}_n(i; b) = 1$. This follows from the definition of z_b^* and the fact that $|M_{i,j}| \geq \tau$. Similarly, when $\textsc{GoodEntry}_{b+L}$ occurs for $b \in [L]$, we have that $|Y_{u,v,b+L}| \geq \tau/2$ if and only if $\textsc{Encode}_n(j; b) = 1$.

Therefore, if at least a $(1 - \lambda)$ fraction of each set of events $\{\textsc{GoodEntry}_b\}_{b \in [L]}$ and $\{\textsc{GoodEntry}_{b+L}\}_{b \in [L]}$ occurs, the decoder calls are guaranteed by Fact 3.3.1 to return (i, j).

Each $\textsc{GoodEntry}_b$ happens with probability at least $(1 - \lambda/4)$, so the expected number of failed events in each set is at most $\lambda L/4$. By Markov's inequality, the chance of having more than λL failures in a set is at most $\frac{1}{4}$. So, conditioned on NoCollide, we have that (i, j) is in the output with probability at least $\frac{1}{2}$.

Then in general, without conditioning, we have that the chance it is included is at least $\frac{1}{4}$, as required. □

So we showed that if we can approximately reconstruct a random grid sketch of the correlation matrix, then we find any one given outlier with probability at least $\frac{1}{4}$. Next we will demonstrate that this approximate reconstruction can be done from row sketches. We end up introducing some small additional error due to the approximation of the rescaling factors, but we will be able to absorb this into the parameters τ and R for our final theorem, without changing the asymptotics.

Lemma 3.3.5. *For any number g of groups which divides n, there is a randomised turnstile matrix streaming algorithm which, given independently and uniformly random sketch maps $\mathcal{B} = (\Gamma_x, \Gamma_y, \sigma_x, \sigma_y)$ compatible with Definition 3.3.3, returns a size $(g \times g \times 2L)$ array W such that for any given $u, v \in [g]^2$ and $b \in [2L]$ we have that:*

$$W_{u,v,b} = \textsc{GridSketch}_g^n(M'; \mathcal{B}; u, v, b) \pm \frac{\tau}{4}$$

with probability at least $(1 - \lambda/8)$, and where M' is an n-by-n matrix where each entry is a $5/4$-approximation of the corresponding entry of $\operatorname{corr}(X)$. Recalling L and λ, the constants from Fact 3.3.1.
This algorithm:

- *Uses $O(g^2 \log n + n^3/\tau^2 g^2)$ bits of space.*
- *Has a query time of $O(\log n(n^3/\tau^2 g^2 + \mathcal{M}(g, O(n^2/\tau^2 g^2))))$.*

where $\mathcal{M}(n, m)$ is the minimum time required to multiply an n-by-m matrix by an m-by-n matrix.

Proof. The initial stages of this process look a lot like the upper bound of Theorem 3.2.1.
Let $\epsilon = \tau\lambda^{\frac{1}{2}} g/68n$ and choose a random $(\epsilon, \lambda/36)$-AMS sketch function S and another random $(1/10, 1/2n^2)$-AMS sketch function S'.

Streaming Phase Maintain sketches $S(r^i)$ and $S'(r^i)$ of each row r^i of X. We also track the sum of all the columns $d \cdot \mu(X)$. This requires $O(n^3/\tau^2 g^2)$ bits of space.

Normalisation At the conclusion of the stream, we can calculate the exact average column from the summed column, and use it to shift every sketch $S(r^i)$ and $S'(r^i)$ to be sketches of the shifted 0-mean version of the row $S(c^i)$ and $S'(c^i)$.
Next, we estimate the variance $\mathbb{V}\left[r^i\right]$ of each row from the norm of $S'(c^i)$. By a union bound with the AMS sketch guarantee, we $10/11$-approximate every variance simultaneously with probability at least $(1 - 1/2n)$.
Using these variances, we approximately normalise the sketches $S(c^i)$ to get sketches of the approximately normalised rows $S(a^i)$. Each a^i has norm between $\frac{11}{10}$ and $\frac{9}{10}$. Let X' be the alternative observation matrix consisting of rows a^i.
Let $M' = \operatorname{corr}(X')$ and observe that each entry is exactly an entry of $\operatorname{corr}(X)$ but distorted proportional to the corresponding variance errors. In particular, each entry is a $\left(\frac{10}{9}\right)^2 \leq \frac{5}{4}$-approximation.
From now on, we will treat it like we have exactly rescaled sketches for the input X'.

Single Grid Sketch Entry Approximation Choose any $u, v \in [g]$ and $b \in [L]$. Let:

$$x = \sum_{i \in \Gamma_x(u)} \sigma_x(i) \cdot S(a^i) \cdot \textsc{Encode}_n(i;\, b)$$
$$y = \sum_{j \in \Gamma_y(v)} \sigma_y(j) \cdot S(a^j)$$

Then consider the inner product query between sketches x and y. Clearly we have:

$$x \odot y = \textsc{GridSketch}_g^n(M';\, \mathcal{B};\, u, v, b) \pm \epsilon \|x\| \|y\|$$

So we need to see that $\epsilon\|x\|\|y\| \leq \tau/4$ with probability at least $(1 - \lambda/8)$.
Both x and y are a sum of vectors, each of norm at most $5/4$, and each multipled by a an independent random sign. Label each of the terms of x as x_i:

$$\begin{aligned}
\left\| \sum_{i \in [n/g]} x_i \right\|^2 &= \left\langle \sum_{i \in [n/g]} x_i, \sum_{i \in [n/g]} x_i \right\rangle \\
&= \sum_{i \in [n/g]} \sum_{j \in [n/g]} \langle x_i, x_j \rangle \\
&= \sum_{i \in [n/g]} \|x_i\|^2 + \sum_{i \in [n/g]} \sum_{j \in [i-1]} 2 \langle x_i, x_j \rangle \\
&\leq \frac{25n}{16g} + \sum_{i \in [n/g]} \sum_{j \in [i-1]} 2 \langle x_i, x_j \rangle
\end{aligned}$$

but this final sum of inner products is a sum of fewer than $(n/g)^2$ uncorrelated terms, each of variance at most $25/4$. So the sum has mean 0 and variance at most $(5n/2g)^2$. By Chebychev's inequality, we have that:

$$\|x\|^2 \leq \left(\frac{25}{16} + \frac{15}{\lambda^{\frac{1}{2}}} \right) \frac{n}{g} \leq \frac{17n}{\lambda^{\frac{1}{2}} g}$$

with probability at least $(1 - \lambda/36)$. A similar argument applies to $\|y\|^2$.
The sketch approximation follows the AMS guarantee with probability at least $(1 - \lambda/36)$, so we have that the sketch approximation is good and the norms are bounded all at the same time with probability at least $(1 - \lambda/8)$, giving:

$$\epsilon\|x\|\|y\| \leq \tau/4$$

as required.
By symmetry, the same process works for $b \in [2L] \setminus [L]$.

Fast All-Entries Approximation All that remains to be seen is that all the necessary inner product can be computer for every $W_{u,v,b}$ we wish to approximate. For each $b \in [2L]$, we exclude the appropriate rows according to $\textsc{Encode}_n(i;\, b)$

or $\textsc{Encode}_n(j; b - L)$, then sum all the row sketches together for each group $\Gamma_x(u)$ and $\Gamma_x(v)$, for $u, v \in [g]$. This requires $O(n \log n/\epsilon^2)$ time to perform all the encoding and summing.

Next, we recall the structure of the AMS sketch. The exact query process is to take the median over $O(\log(1/\delta))$ vectors each of size $O(1/\epsilon^2)$. Each batch of inner products can be done simultaneously between all groups as the multiplication of an g-by-$O(1/\epsilon^2)$ matrix with an $O(1/\epsilon^2)$-by-g matrix. Using fast matrix multiplication, this can take significantly less time than the naive approach.

The total running time to perform all $O(\log n)$ inner products is then $O(\log n \mathcal{M}(g, O(1/\epsilon^2)))$, and we require $O(g^2 \log n)$ bits of space to store the resulting W. □

Theorem 3.3.6. *There is a $1/n$-error randomised turnstile matrix streaming algorithm for finding all the indices in $\mathrm{OUT}(M, \tau)$ which:*

- *Uses $O(n^{\frac{5}{3}}/\tau^2)$ bits of space.*
- *Has a query time of $O(n^{\frac{5}{3}} \log^2 n/\tau^2)$.*

as long as $\mathrm{RES}(M, \tau) \leq \tau^2 \lambda n^{\frac{4}{3}}/256$ and $|\mathrm{OUT}(M, \tau)| \leq n^{\frac{2}{3}}/4$.

Proof. Let $g = n^{\frac{2}{3}}$.

We simply run $T = O(\log n)$ copies of the algorithm from Lemma 3.3.5 and then use Lemma 3.3.4 on each approximately generated grid sketch. This gives us T independent chances to find each $(i, j) \in \mathrm{OUT}(M, \tau)$ with probability at least $\frac{1}{4}$. So we find all of them at least once with probability at least $(1 - 1/n)$.

Then the costs follow from the fact that we can multiply an $n^{\frac{2}{3}}$-by-$O(n^{\frac{2}{3}}/\tau^2)$ matrix with an $O(n^{\frac{2}{3}}/\tau^2)$-by-$n^{\frac{2}{3}}$ matrix by performing $O(1/\tau^2)$ square matrix multiplications of width $n^{\frac{2}{3}}$. The current best bound on the exponent of matrix multiplication stands at $\omega < 2.3728639 < 2.4$ due to Gall [Gal14], so this takes at most $O(n^{\frac{2\omega}{3}}/\tau^2) \leq O(n^{\frac{5}{3}}/\tau^2)$ time. □

4 Independent Sets

4.1 Introduction

4.1.1 Background

In this chapter we examine questions about independent sets in graph streams. For a given graph, an independent set is simply any subset of the vertices which induces a graph with no edges. Independent sets play a fundamental role in graph theory, with many applications to optimisation and scheduling problems. Of particular interest are *maximal* independent sets and *maximum* independent sets.

Definition 4.1.1 (Independent Sets)**.** *Consider a graph $G = (V, E)$:*

- *An **independent set** in G is a subset of vertices $I \subset V$ such that: for every pair of vertices $v_1, v_2 \in I$ there is no edge between them in the graph $\{v_1, v_2\} \notin E$.*
- *A **maximal independent set** in G is any independent set which is not a strict subset of another independent set in G.*
- *Let $\alpha(G)$ be the maximum size $|I|$ over independent sets I in G.*
- *A **maximum independent set** in G is any independent set of size $\alpha(G)$.*
- *A **c-approximate maximum independent set** in G is any independent set of size at least $\alpha(G)/c$.*

Maximal Independent Set

Finding a maximal independent set is quite straightforward: start from an empty independent set and then iterate through the vertices in an arbitrary order. Greedily add each vertex to the set as long as doing so would not violate the independent set constraint. At the end, every vertex was either added or could not be added without violating the constraint - hence we have a maximal independent set. Call this strategy GreedyIS.

This GreedyIS algorithm is straightforward to implement on vertex streams (both implicit and explicit), using only as much space as required to store the output.

Fact 4.1.2. *The* GreedyIS *algorithm for independent sets is a one-pass* $\mathrm{O}(\alpha(G) \cdot \log n) \leq \mathrm{O}(n \log n)$ *bits of space maximal independent set algorithm in explicit vertex streams, and* $\mathrm{O}(\alpha(G) \log |\mathcal{U}|)$ *bits of space in implicit vertex streams.*

We simply keep track of the chosen vertices as a set of vertex labels (for explicit vertex streams) or a multi-set of vertex identifiers (for implicit vertex streams), trying to add each vertex as it arrives in stream order.

Maximum Independent Set

The problem of finding a maximum independent set of a graph is very challenging in general. For arbitrary graphs it is well known as one of the early NP-Hard problems [Kar72], and it is NP-Hard to even $n^{1-\epsilon}$-approximate for any $\epsilon > 0$ [Hås96; Zuc07].

However, this does not immediately make space-efficient streaming algorithms impossible. An $\tilde{\mathrm{O}}(n)$ bit algorithm would give us a very meaningful space saving but could still require exponential $\Omega(2^n)$ time to process the whole stream.

Halldórsson, Sun, Szegedy, and Wang [Hal+12] considered the related problem of approximating the size of the largest clique in an insert-only edge stream.

Definition 4.1.3 (Cliques)**.** *Consider a graph* $G = (V, E)$*:*

- *A* ***clique*** *in* G *is a subset of vertices* $K \subset V$ *such that: every pair of vertices* $v_1, v_2 \in K$ *are adjacent in the graph* $\{v_1, v_2\} \in E$*.*
- *Let* $\omega(G)$ *be the maximum size* $|K|$ *over cliques* K *in* G*.*

Their result shows that at least $\Omega\left(n^2/c^2 \log^2 n\right)$ bits of space must be used for any c-approximation algorithm for $\omega(G)$, rising to $\Omega\left(n^2/c^4\right)$ bits when $c = o(\log n)$. In particular, any constant-factor approximation must store $\Omega(n^2)$ bits, essentially the whole graph.

They also show a corresponding tight (up to log factors) upper bound: a $\frac{1}{3}$-error randomised edge streaming algorithm for c-approximating maximum clique using $\tilde{\mathrm{O}}(n^2/c^2)$ bits of space.

Their bounds apply equally well to the problem of c-approximating $\alpha(G)$, giving us a tight $\tilde{\Theta}(n^2/c^2)$ bound on the edge streaming complexity.

However, their construction does not apply to vertex streams. While the two-party construction they used is enough to show hardness for edge streams, any communication lower bound for vertex streams must use more than c parties, otherwise we can only deduce a trivial lower bound of zero bits.

Fact 4.1.4. *Let $k \leq c$. Suppose we split a vertex stream into contiguous pieces among k parties.*
Without any communication, at least one party knows a c-approximation to maximum independent set in the streamed graph.

Proof. Each party holds an induced subgraph of the full graph.
Consider a maximum independent set I of the whole graph G. At least one player must hold at least $|I|/c$ vertex of I, so that player can find an independent set of size at least $\alpha(G)/c$.
Since they have an induced subgraph, they also know an independent set of size $\alpha(G)/c$ of the whole graph. So one of the players already knows the answer before any communication has even happened. □

Special Graph Classes

Despite negative results for arbitrary graphs in both the offline and edge streaming settings, there are many positive results known for special classes of graphs. In the offline world, many classes of graphs have polynomial time exact algorithms or polynomial time approximation schemes.

In the streaming setting, the only special class algorithms we are aware of are for interval intersection graphs. Emek, Halldórsson, and Rosén [EHR16] considered the problem of finding an approximately largest subset of non-overlapping intervals from a stream of intervals. For unit or proper intervals, they showed an efficient 3/2-approximation, while for arbitrary intervals they managed a 2-approximation. Complementary lower bounds show that no better approximation quality is possible without storing essentially the whole stream.

Later Cabello and Pérez-Lantero [CPL17] extended these techniques to allow estimation of $\alpha(G)$ in polylogarithmic space with only an arbitrarily small ϵ loss in approximation quality. They use clever space partitioning techniques with a sampling strategy.

4.1.2 Results

These three tables list the main complexity bounds proven in this chapter. Unless otherwise specified, the bounds are on the space cost of the most efficient randomised $\frac{1}{3}$-error streaming algorithm for that problem.

Edge Streaming Problems	Complexity	Theorems
Find a maximal independent set	$\Omega(n^2)$	4.3.1
Find a $\frac{24}{25}$-maximal independent set	$n^{1+\Omega(1/\log\log n)}$	4.3.7
Approximate $\beta(G)$ within factor c	$\tilde{\Theta}(n/c)$	4.4.5, 4.4.6
Return any x satisfying $\alpha(G) \geq x \geq \beta(G)/c$	$\tilde{\Theta}(n/c)$	4.4.14

Explicit Vertex Streaming Problems	Complexity	Theorems
Approximate $\alpha(G)$ or $\omega(G)$ within factor c	$\Omega(n^2/c^6)$	4.3.9, 4.3.10
Approximate $\chi(G)$ within factor c	$\Omega(n^2/c^6)$	4.3.13
Approximate $\beta(G)$ within factor c	$\tilde{\Theta}(n/c)$	4.4.5, 4.4.6
Return any x satisfying $\alpha(G) \geq x \geq \beta(G)/8\log n$	$O(\log^3 n)$	4.4.12
Approximate $\alpha(G)$ of a unit interval graph within factor $\left(\frac{5}{3}-\epsilon\right)$ for $\epsilon > 0$	$\Omega(n)$	4.5.1

Implicit Vertex Streaming Problems	Complexity	Theorems
Approximate $\alpha(G)$ within factor c	$\tilde{\mathrm{O}}\left(n/c \cdot \log\lvert\mathcal{U}\rvert\right)$	4.3.14
Approximate $\alpha(G)$ within factor c	$\tilde{\Omega}\left(n/c^2 \cdot \log\lvert\mathcal{U}\rvert\right)$	4.3.15
Deterministically find a 3-approximate maximum independent set of a unit square graph	$\tilde{\mathrm{O}}(\alpha(G))$	4.5.2
Approximate $\alpha(G)$ of a unit square graph within factor $(3+\epsilon)$ for $\epsilon > 0$	$\tilde{\mathrm{O}}(\epsilon^{-2})$	4.5.3
Approximate $\alpha(G)$ of a unit square graph within factor $\left(\frac{5}{2}-\epsilon\right)$ for $\epsilon > 0$	$\Omega(n)$	4.5.5
Approximate $\alpha(G)$ of a square graph within factor $(3-\epsilon)$ for $\epsilon > 0$	$\Omega(n)$	4.5.6

4.2 Chained Index Communication Problem

Before we consider any of our questions about independent set streaming problems, we need to define and prove hardness of a new communication problem which we will make use of in several of our new streaming lower bounds.

4.2.1 Problem Definition

We define a one-way multi-party communication problem $\textsc{Chain}_n^k$. The problem is closely related to pointer jumping and generalizes the classic two-party $\textsc{Index}_n$ communication problem to more parties by "chaining" together multiple instances which have the same answer but are otherwise independent.

In $\textsc{Chain}_n^k$, each party (except the last) holds a binary vector that contains a special

bit which is the answer to the instance. Each party (except the first) knows where the answer bit is located in the previous party's vector. Communication is one-way and private, with each player receiving a message from the previous player and then sending a message to the next player. Formally:

Definition 4.2.1 (Chained Index)**.** *The k-party* ***chained index problem*** $\textsc{Chain}_n^k$ *consists of:*

- $(k-1)$ *n-bit binary vectors* $\{X^{(i)}\}_{i=1}^{k-1}$
- *Corresponding indices* $\{\sigma_i\}_{i=1}^{k-1}$ *from the range* $[n]$

We know there is an answer bit $z \in \{0,1\}$. We are promised that for every $i \in [k-1]$, the entry $X_{\sigma_i}^{(i)} = z$.

The input is initially allocated as follows:

- *The first party P_1 knows $X^{(1)}$*
- *Each intermediate party P_i for $1 < i < k$ knows $X^{(i)}$ and σ_{i-1}*
- *The final party P_k knows just σ_{k-1}*

Communication is one-way with P_1 sending a message to P_2, then P_2 to P_3 and so on. After all messages are sent, P_k must correctly output z, succeeding with probability at least $\frac{2}{3}$. If the promise condition is violated, any output is considered correct.

There is a trivial communication upper bound of $O(n)$ bits: for instance, simply have the penultimate party send $X^{(k-1)}$ to the final party who can then return $X_{\sigma_{k-1}}^{(k-1)}$.

We claim two lower bounds on the communication complexity of this problem.

Theorem 4.2.2. *Any communication scheme $\mathcal{B}$ which solves $\textsc{Chain}_n^k$ must communicate at least $\Omega\left(\frac{n}{k^2}\right)$ bits in total.*

This first bound is shown by a simple reduction from instances of another problem (conservative one-way Boolean pointer jumping [Cha07]) to instances of our problem. We will prove this in Section 4.2.2.

Theorem 4.2.3. *There is a constant $C > 0$ such that: any communication scheme $\mathcal{B}$ which solves $\textsc{Chain}_n^k$ for $k \leq C\left(\frac{n}{\log n}\right)^{\frac{1}{4}}$ must communicate at least $\Omega\left(\frac{n}{k}\right)$ bits in total.*

This second bound is shown by a much more involved proof, though still closely based on the structure of the pointer jumping bound given in [Cha07]. We will prove this in Section 4.2.2.

In particular, for constant k, we have a tight bound on the communication complexity

of the k-party chained index problem of $\Theta(n)$. On the upper bound side, it seems difficult to gain any advantage as the number of parties grows, so we conjecture that a dependence on k is not necessary.

Conjecture 4.2.4. *Any communication scheme for* $\textsc{Chain}_n^k$ *requires* $\Omega(n)$ *bits of communication in total.*

4.2.2 Reduction from Conservative One-Way Pointer Jumping

Consider the following problem.

Definition 4.2.5 (Point Jumping)**.** *The k-party* ***conservative one-way Boolean pointer jumping problem*** $\textsc{Jump}_n^k$ *consists of a starting index $\alpha \in [n]$ and $k-1$ functions $\{f_i\}_{i=2}^k$.*
The first $k-2$ are of the form $f_i : [n] \to [n]$, and the final one is of the form $f_{k-1} : [n] \to \{0,1\}$. We use $f_{i:j}$ to refer to the composition of functions $f_i \circ f_{i+1} \circ \cdots \circ f_{j-1} \circ f_j$, using the convention that $(g \circ h)(x) = h(g(x))$.

The input is divided as follows:

- *The first party P_1 knows all the functions $\{f_i\}_{i=2}^k$*
- *The second party P_2 knows α and every f_j for $j \geq 3$*
- *Each other party P_i knows $f_{2:i-1}(\alpha)$ and every f_j for $j \geq i+1$*

Each party sends exactly one message in ascending order to their immediate successor: P_1 sends to P_2, then P_2 sends to P_3, and so on. After all messages are sent, P_k must correctly output $f_{2:k}(\alpha)$ with probability at least $\frac{2}{3}$.

The conservative version of one-way k-party pointer jumping problem was introduced and studied by Damm and Jukner [DJS98], who showed a communication lower bound of $\Omega(\frac{n}{k^2})$ for $k \in o(n^{\frac{1}{3}})$ for a version of this problem with non-Boolean final layer. Later, Chakrabarti [Cha07] extended this to all k and to the Boolean version.

These papers both consider a more general blackboard communication model, where all parties can see all messages, while we only need to allow private messages for our streaming bounds. The private message case is always at least as hard, as any private message protocol also works as a blackboard protocol.

Theorem 4.2.6 (Theorem 2 in [Cha07])**.** *Any communication scheme $\mathcal{A}$ which solves* $\textsc{Jump}_n^k$ *must communicate at least $\Omega(\frac{n}{k^2})$ bits.*

The communication hardness for our new problem is then as follows.

Theorem 4.2.2 (restated)**.** *Any communication scheme $\mathcal{B}$ which solves* $\textsc{Chain}_n^k$ *must communicate at least* $\Omega\left(\frac{n}{k^2}\right)$ *bits in total.*

Proof. We prove the claim by showing that any instance of $\textsc{Jump}_n^k$ can be reduced to an instance of $\textsc{Chain}_n^k$ without any communication. Hence, any algorithm which solves $\textsc{Chain}_n^k$ can solve $\textsc{Jump}_n^k$ with no change in the communication cost. Combining this with the lower bound of Theorem 4.2.6 gives the result.

Reduction Fix an instance of $\textsc{Jump}_n^k$. For each i let $X^{(i)}$ be the binary vector whose j^{th} entry is $f_{i+1:k}(j)$. For each i let $\sigma_i = f_{2:i}(\alpha)$. Now we observe three facts:

- Every $\{X^{(i)}_{\sigma_i}\}_{i=1}^{k-1}$ is equal to $f_{2:k}(\alpha)$
- Each party P_i for $i < k$ knows all the information required to compute $X^{(i)}$
- Each party P_i for $i > 1$ knows all the information required to compute σ_{i-1}

So we have constructed (with no communication) a k-party chained index problem which, if solved, will tell us exactly $f_{2:k}(\alpha)$. It therefore follows that the communication cost for any solution to $\textsc{Chain}_k$ is at least that for $\textsc{Jump}_k$. □

4.2.3 Towards an Improved Lower Bound

In this section, we prove a few key properties that we need for the proof of Theorem 4.2.3. The proof itself is in the subsequent Section 4.2.4.

Lemma 4.2.7. *Consider a uniformly random pointer $\sigma \in [n]$, an independent uniformly random bit $B \in \{0,1\}$, and a length n vector X consisting of independent uniformly random bits, except X_σ is set to be equal to B. Then we have:*

$$I(B : X) \leq O\left(\sqrt{\frac{\log n}{n}}\right)$$

Proof. Observe that for every b and x: $\mathbb{P}_B(b) = \frac{1}{2}$, $\mathbb{P}_X(x) = \frac{1}{2^n}$, and $\mathbb{P}_B(b \,|\, X = x) = \frac{1}{n}|x|_b$ where $|x|_b$ is the number of entries in x which match bit b.

Combined with the definition of mutual information, we can see that:

$$\begin{aligned} I(x:X) &= \sum_{x\in\{0,1\}^n}\sum_{b\in\{0,1\}} \mathbb{P}_{B,X}(b,x)\log\left(\frac{\mathbb{P}_{B,X}(b,x)}{\mathbb{P}_B(b)\cdot\mathbb{P}_X(x)}\right) \\ &= \sum_{Z\in\{0,1\}^n}\sum_{z\in\{0,1\}} \mathbb{P}_B(b\,|\,X=x)\cdot\mathbb{P}_X(x)\cdot\log\left(2\cdot\mathbb{P}_B(b\,|\,X=x)\right) \\ &= \frac{1}{2^n}\sum_{Z\in\{0,1\}^n}\sum_{z\in\{0,1\}} \mathbb{P}_B(b\,|\,X=x)\cdot\log\left(2\cdot\mathbb{P}_B(b\,|\,X=x)\right) \\ &= \frac{1}{2^n}\cdot\sum_{Z\in\{0,1\}^n}\left(\frac{|Z|_0}{n}\log\left(\frac{2|Z|_0}{n}\right)+\frac{|Z|_1}{n}\log\left(\frac{2|Z|_1}{n}\right)\right) \end{aligned}$$

We now have a sum over binary words which only depends on the number of zero (and non-zero) entries of those words. By symmetry of the binomial distribution (in terms of $|Z|_0$ vs $|Z|_1$) we only need to consider $|Z|_0$. We can also rewrite the sum in terms of $k = |x|_0$ and include the binomial coefficient $\binom{n}{k}$.

$$I(x:X) = \frac{1}{2^n}\cdot\sum_{k\in[n]}\binom{n}{k}\left(\frac{2k}{n}\log\left(\frac{2k}{n}\right)\right)$$

We wish to re-express the log factor in the form $\log(1+\delta)$ to allow us to use a polynomial approximation. To guarantee convergence, we first split the sum between k values close to $\frac{n}{2}$ and the rest.
Consider $\mathcal{N}\subset[n]$, the set of integers within $\pm\sqrt{n\log n}$ of the mean number of 0's $\frac{n}{2}$. By applying Hoeffding's inequality to the appropriate Binomial distribution, we can see that the number of vectors $x\in\{0,1\}^n$ such that $|x|_0\in[n]\setminus\mathcal{N}$ is bounded above by $2^n\exp(-2\log n) = \frac{2^n}{n^2}$.
Then:

$$\begin{aligned} I(B:X) &\le \frac{1}{2^n}\left(\sum_{k\in\mathcal{N}}\binom{n}{k}\frac{2k}{n}\log\left(\frac{2k}{n}\right)+\sum_{k\in[n]\setminus\mathcal{N}}\binom{n}{k}2\log(2)\right) \\ &\le \frac{1}{2^n}\left(\sum_{k\in\mathcal{N}}\binom{n}{k}\frac{2k}{n}\log\left(\frac{2k}{n}\right)\right)+O\left(\frac{1}{n^2}\right) \end{aligned}$$

Now since for the remaining values of k we have $\delta = 1-\frac{2k}{n}$ bounded by $\pm2\sqrt{\frac{\log n}{n}}$, as long as n is not too small we can use the first order Taylor approximation

$\log(1+\delta) = O(\delta)$:

$$\begin{aligned} I(B:X) &\leq \frac{1}{2^n}\left(\sum_{k\in\mathcal{N}}\binom{n}{k}\frac{2k}{n}O(\delta)\right) + O\left(\frac{1}{n^2}\right) \\ &\leq \frac{1}{2^n}\left(\sum_{k\in\mathcal{N}}\binom{n}{k}\frac{2k}{n}O\left(\sqrt{\frac{\log n}{n}}\right)\right) + O\left(\frac{1}{n^2}\right) \\ &= \frac{1}{2^n}\left(\sum_{k\in\mathcal{N}}2\binom{n-1}{k-1}O\left(\sqrt{\frac{\log n}{n}}\right)\right) + O\left(\frac{1}{n^2}\right) \\ &\leq O\left(\sqrt{\frac{\log n}{n}}\right) + O\left(\frac{1}{n^2}\right) \leq O\left(\sqrt{\frac{\log n}{n}}\right) \end{aligned}$$

□

A tighter $O\left(\frac{1}{n}\right)$ bound for Lemma 4.2.7 seems achievable by using a higher order Taylor approximation for log, and using combinatorial identities to perform the sum. However, this would not lead to any improvement of Theorem 4.2.3 without a corresponding tightening of Lemma 4.2.9.

Lemma 4.2.8. *Consider a pair of random variables X and Y over the same range $\mathcal{Z}$. Suppose that there is some $\delta \in (0,1)$ such that for every $z \in \mathcal{Z}$ we have:*

$$(1-\delta)\cdot\mathbb{P}_Y(z) \leq \mathbb{P}_X(z) \leq (1+\delta)\cdot\mathbb{P}_Y(z)$$

then:

$$H(X) \leq (1+\delta)\cdot H(Y) + O(\delta)$$

Proof. Recall that

$$\begin{aligned} H(X) &= \sum_{z\in\mathcal{Z}}\mathbb{P}_X(z)\log\frac{1}{\mathbb{P}_X(z)} \\ &\leq \sum_{z\in\mathcal{Z}}(1+\delta)\mathbb{P}_Y(z)\log\frac{1}{(1-\delta)\cdot\mathbb{P}_Y(z)} \\ &= \sum_{z\in\mathcal{Z}}(1+\delta)\mathbb{P}_Y(z)\log\frac{1}{\mathbb{P}_Y(z)} + (1+\delta)\mathbb{P}_Y(z)\log\frac{1}{1-\delta} \\ &\leq (1+\delta)H(Y) + (1+\delta)\log\frac{1}{1-\delta} \end{aligned}$$

Then, since $|\delta| < 1$, we can use the Taylor expansion of $-\log(1-\delta)$ to bound the second term by $O(\delta)$. □

Lemma 4.2.9. *Consider a pair independent uniform pointer $A, B \in [n]$ and a pair of length n vectors X, Y consisting of independent uniformly random bits, except Y_B is set to be equal to X_A. Suppose we have a function f. Then we have, for every $i \in [n]$:*

$$I(X_i : f(X) \,|\, Y, A = i) \leq I(X_i : f(X) \,|\, Y, A) + O\left(\sqrt{\frac{\log n}{n}}\right)$$

Proof. From the definition of conditional mutual information, we have that the difference:

$$\begin{aligned} d &= I(X_i : f(X) \,|\, Y, A = i) - I(X_i : f(X) \,|\, Y, A) \\ &= \frac{1}{n} \sum_{a \in [n]} \frac{1}{2^n} \sum_{y \in \{0,1\}^n} (I(X_i : f(X) \,|\, Y = y, A = i) - I(X_i : f(X) \,|\, Y = y, A = a)) \end{aligned}$$

To try to keep things succinct, we use the shorthand:

- $\mathcal{I}_{a,y} = I(X_i : f(X) \,|\, Y = y, A = a)$
- $d_a = \frac{1}{2^n} \sum_{y \in \{0,1\}^n} (\mathcal{I}_{i,y} - \mathcal{I}_{a,y})$

then we can rewrite d as the average of differences $d = \frac{1}{n} \sum_{a \in [n]} d_a$, so we just need to be able to bound each d_a.

Consider d_a for fixed $a \in [n] \setminus \{i\}$ (when $a = i$, we clearly have $d_a = 0$). Let $\mathcal{S} \subset \{0,1\}^n$ be the set of vectors y with number of zeros $|y|_0$ within $\pm\sqrt{n \log n}$ of the mean $\frac{n}{2}$. Much the same is in the proof of Lemma 4.2.7, we can use Hoeffding's inequality on a Binomial distribution to see that $|\{0,1\}^n \setminus \mathcal{S}| \leq \frac{2^n}{n^2}$.

We now split the sum d_a into two parts according to $\mathcal{S}$ and its complement. Since the mutual information terms are always bounded $\mathcal{I}_{a,y} \leq H(X_i \,|\, Y = y, A = a) \leq 1$, we can see that:

$$\begin{aligned} d_a &= \frac{1}{2^n} \sum_{y \in \mathcal{S}} (\mathcal{I}_{i,y} - \mathcal{I}_{a,y}) + \frac{1}{2^n} \sum_{y \in \{0,1\}^n \setminus \mathcal{S}} (\mathcal{I}_{i,y} - \mathcal{I}_{a,y}) \\ &\leq \frac{1}{2^n} \sum_{y \in \mathcal{S}} (\mathcal{I}_{i,y} - \mathcal{I}_{a,y}) + \frac{|\{0,1\}^n \setminus \mathcal{S}|}{2^n} \cdot (2) \\ &\leq \frac{1}{2^n} \sum_{y \in \mathcal{S}} (\mathcal{I}_{i,y} - \mathcal{I}_{a,y}) + O\left(\frac{1}{n^2}\right) \end{aligned}$$

Now for fixed $y \in \mathcal{S}$, we consider the term $(\mathcal{I}_{i,y} - \mathcal{I}_{a,y})$. We just need to be able to bound each of these to complete the result.

Recall from the definition of mutual information that we can say that

$$\mathcal{I}_{j,y} = H(X_i \,|\, Y = y, A = j) - H(X_i \,|\, Y = y, A = j, f(X))$$

hence

$$\begin{aligned} \mathcal{I}_{i,y} - \mathcal{I}_{a,y} = &(H(X_i \,|\, Y = y, A = i) - H(X_i \,|\, Y = y, A = a)) \\ &+ (H(X_i \,|\, Y = y, A = a, f(X)) - H(X_i \,|\, Y = y, A = i, f(X))) \quad \text{(I)} \end{aligned}$$

We would like to bound both of these entropy differences using Lemma 4.2.8.
First, we set some shorthand to compare the probabilities. For any $j \in [n]$, let:

- $p_j(*, g) = \mathbb{P}_{f(X)}(g \mid Y = y, A = j)$
- $p_j(x_i, *) = \mathbb{P}_{X_i}(x_i \mid Y = y, A = j)$
- $p_j(x_i, g) = \mathbb{P}_{X_i, f(X)}(x_i, g \mid Y = y, A = j)$
- $p_j(x_i \mid g) = \mathbb{P}_{X_i}(x_i \mid f(X) = g, Y = y, A = j)$

For the first entropy difference in equation (I), we need to consider the relationship between the probability distribution $p_i(x_i)$ and $p_a(x_i)$.
Clearly, $p_a(x_i) = \frac{1}{2}$ (since $a \neq i$) as we are conditioning on things independent of the random bit. However, $p_i(0) = \frac{|y|_0}{n}$ and $p_{y,i}(1) = \frac{|y|_1}{n}$ where $|y|_0$ and $|y|_1$ are the number of 0's and 1's (respectively) in vector y. Since $y \in \mathcal{S}$, we know that $|y|_0, |y|_1 \in \frac{n}{2} \pm \sqrt{n \log n}$.
Hence, we get that $p_i(x_i) \leq \left(1 + O\left(\sqrt{\frac{\log n}{n}}\right)\right) \cdot p_a(x_i)$ for use in Lemma 4.2.8.
Recalling also that $H(X_i \mid Y = y, A = a) \leq 1$, we then have:

$$\begin{aligned}
\mathcal{I}_{i,y} - \mathcal{I}_{a,y} &\leq O\left(\sqrt{\frac{\log n}{n}}\right) \cdot (H(X_i \mid Y = y, A = a)) + O\left(\sqrt{\frac{\log n}{n}}\right) \\
&\quad + (H(X_i \mid Y = y, A = a, f(X)) - H(X_i \mid Y = y, A = i, f(X))) \\
&\leq O\left(\sqrt{\frac{\log n}{n}}\right) \\
&\quad + (H(X_i \mid Y = y, A = a, f(X)) - H(X_i \mid Y = y, A = i, f(X))) \quad \text{(II)}
\end{aligned}$$

Now for the second entropy difference, we recall the definition of conditional entropy, and restate it as:

$$\sum_g \mathbb{P}_{f(X)}(g) \cdot \Big(H(X_i \mid Y = y, A = a, f(X) = g) - H(X_i \mid Y = y, A = i, f(X) = g)\Big) \quad \text{(III)}$$

so we can bound it by bounding the difference for each $f(X) = g$.
Fix g and x_i. For each $x_a \in \{0, 1\}$, consider the set

$$S_{x_a} = \{X \in \{0,1\}^n \mid X_i = x_i, X_a = x_a, f(X) = g\}$$

We can express the joint probabilities of g and x_i as follows:

$$p_i(x_i, g) = \left(\frac{|y|_{x_i}}{n}\right) \cdot \left(\frac{|S_0| + |S_1|}{2^{n-1}}\right) = \frac{|S_0||y|_{x_i} + |S_1||y|_{x_i}}{n2^{n-1}}$$

$$p_a(x_i, g) = \left(\frac{1}{2}\right) \cdot \left(\frac{|S_0| \cdot |y|_0 + |S_1| \cdot |y|_1}{n2^{n-2}}\right) = \frac{|S_0||y|_0 + |S_1||y|_1}{n2^{n-1}}$$

Hence we have the following bound for both probabilities:

$$\frac{|S_0| + |S_1|}{n2^{n-1}} \cdot \min\{|y|_0, |y|_1\} \leq p_i(x_i, g), p_a(x_i, g) \leq \frac{|S_0| + |S_1|}{n2^{n-1}} \cdot \max\{|y|_0, |y|_1\}$$

In particular, this means that

$$\frac{\min\{|y|_0,|y|_1\}}{\max\{|y|_0,|y|_1\}}\cdot p_a(x_i,g)\le p_i(x_i,g)\le\frac{\max\{|y|_0,|y|_1\}}{\min\{|y|_0,|y|_1\}}\cdot p_a(x_i,g)$$

By linearity, an analogous inequality holds for $p_a(*,g)$ and $p_i(*,g)$.
Now, we see that:

$$p_i(x_i|g)=\frac{p_i(x_i,g)}{p_i(*,g)}\le\left(\frac{\max\{|y|_0,|y|_1\}}{\min\{|y|_0,|y|_1\}}\right)^2\cdot\frac{p_a(x_i,g)}{p_a(*,g)}=\left(\frac{\max\{|y|_0,|y|_1\}}{\min\{|y|_0,|y|_1\}}\right)^2\cdot p_a(x_i|g)$$

Recalling that as $y\in\mathcal{S}$, $\max\{|y|_0,|y|_1\}\le\frac{n}{2}+\sqrt{n\log n}$ and $\min\{|y|_0,|y|_1\}\ge\frac{n}{2}-\sqrt{n\log n}$, we get that

$$p_i(x_i|g)\le\left(1+O\left(\sqrt{\frac{\log n}{n}}\right)\right)\cdot p_a(x_i|g)$$

Applying Lemma 4.2.8 with this bound to each term in equation (III), we get that

$$\mathcal{I}_{i,y}-\mathcal{I}_{a,y}\le O\left(\sqrt{\frac{\log n}{n}}\right)$$

This then gives the same bound when we average over y to get d_a, and then average over a to get d, giving the result.

□

This final lemma is the key to the party-elimination strategy we will use in the proof of Theorem 4.2.3. Roughly speaking, this lemma tells us to what extent we can approximate a function of three related inputs if we are only provided with two of them and must guess the third.

Lemma 4.2.10. *Let A, B, C, be random variables with ranges $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ respectively. Variables A and B are not required to be independent. Then, for every function $f:\mathcal{A}\times\mathcal{B}\times\mathcal{C}\to[0,1]$, there exists a function $g:\mathcal{B}\to\mathcal{C}$ such that*

$$\begin{aligned}\mathbb{E}_{A,B}[f(A,B,g(B)]&\le\mathbb{E}_{A,B,C}[f(A,B,C)]+\sqrt{\frac{\ln 2}{2}\cdot I(A:B,C)}\\&=\mathbb{E}_{A,B,C}[f(A,B,C)]+\sqrt{\frac{\ln 2}{2}\cdot(I(A:C\,|\,B)+I(A:B))}\end{aligned}$$

Proof sketch. Essentially identical to the proof of [Cha07, Lemma 8], except A and B are not promised to be independent. We instead use the chain rule to get:

$$I(A:B,C)=I(A:C\,|\,B)+I(A:B)$$

□

4.2.4 Improved Lower Bound Proof

To show this, we will lean heavily on the machinery and proof structure used by Chakrabarti to prove [Cha07, Theorem 16].

The first thing we need to borrow is the concept of the *information cost* of a multi-party protocol.

Definition 4.2.11 (Information cost)**.** *Suppose we have a distribution of inputs $\mathcal{D}$ for a k-party problem, and a protocol P for solving the problem over that distribution. The information cost of the protocol over the distribution is given by:*

$$\textsc{icost}(P, \mathcal{D}) = I(A : M \mid B)$$

Where A is player 1's input, B is player 2's input, and M is the message sent from player 1 to player 2.

Next, we need to define a related communication problem and a few distributions.

Definition 4.2.12. *Let $\textsc{Toy}_n^k$ be a communication problem on k parties where parties 2 through k have an instance of $\textsc{Chain}_n^{k-1}$ (labelling the $X^{(i)}$'s and σ_i's from 2 to k instead of 1 to $k-1$), and player 1 just has the answer bit $x^* = X^{(2)}_{\sigma_2}$.*
The goal is to send one-way messages to solve the instance of $\textsc{Chain}_n^{k-1}$.

Clearly, this is quite a trivial problem, player 1 can simply pass x^* down the line of players giving an $O(k)$ upper bound. However, we can use it to help us create protocols for $\textsc{Chain}_n^{k-1}$ with the following observation.

Fact 4.2.13. *Any protocol for $\textsc{Toy}_n^k$ where player 1 always sends 0 bits to player 2 can also be used as a protocol for $\textsc{Chain}_n^{k-1}$ (simply have each player i pretend to be player $i+1$).*

This will form the basis of our plan. Take a protocol for $\textsc{Chain}_n^k$ and use it to make a protocol for $\textsc{Toy}_n^k$. Then, tweak the protocol to eliminate the message between player 1 and player 2, thus create a protocol for $\textsc{Chain}_n^{k-1}$. By repeating this process we can get a protocol for $\textsc{Chain}_n^2$, also known as the $\textsc{Index}_n$ problem, which has known hardness.

Definition 4.2.14. *Let $\mathcal{U}^k$ be the uniform distribution over instances of $\textsc{Chain}_n^k$, and let $\mathcal{V}^k$ be the uniform distribution over instances of $\textsc{Toy}_n^k$.*

First we will show how to turn protocols for $\mathcal{U}^k$ into protocols for $\mathcal{V}^k$.

Lemma 4.2.15. *Suppose we have an ϵ-error deterministic protocol P with fixed message lengths $(l_1, \cdots, l_{k-1})$ which can solve problems sampled from $\mathcal{U}^k$.*

For each $i \in [n]$ we can define the private coin random protocol Q_i for instances of $\textsc{Toy}_n^k$ sampled from $\mathcal{V}^k$. In Q_i we set $\sigma_1 = i$ and player 1 generates $n-1$ uniformly random bits to build $X^{(1)}$ (with $X_i^{(1)} = x^*$), players then proceed acting as if they are running protocol P.

Let ϵ_i be the error probability of Q_i on $\mathcal{V}^k$. Then we can show:

1. *Each Q_i has the same message lengths: $(l_1, \cdots, l_{k-1})$*
2. *The average error is the original error: $\frac{1}{n}\sum_{i\in[n]} \epsilon_i = \epsilon$*
3. *The average information cost is bounded by:*

$$\sum_{i\in[n]} \frac{1}{n}\textsc{icost}(Q_i, \mathcal{V}^k) \leq \frac{1}{n}\textsc{icost}(P, \mathcal{U}^k) + O\left(\sqrt{\frac{\log n}{n}}\right) \leq \frac{l_1}{n} + O\left(\sqrt{\frac{\log n}{n}}\right)$$

Proof. For part (1), the possible message transcripts for Q_i are a subset of the possible message transcripts for P, so the message lengths are the same.

For part (2), consider what would happen if we sample a problem from $\mathcal{V}^k$ and then uniformly at random chose a Q_i (out of the possible $i \in [n]$) to run. If we augment the sample from $\mathcal{V}^k$ with the generated $X^{(1)}$ and σ_1, the combined simulated input has the same distribution as $\mathcal{U}^k$, and behaviour of the players looks exactly the same as if we ran P on the corresponding instance. Hence, $\frac{1}{n}\sum_{i\in[n]} \epsilon_i = \epsilon$.

Finally, for part (3) we consider the information cost of each Q_i over $\mathcal{V}^k$. Let $M_i(x^*)$ be the random variable representing the message sent by player 1 in Q_i for input x^*, and use $M(X^{(1)})$ to mean the message sent by player 1 in protocol P for input $X^{(1)}$. Notice that $M(X^{(1)})$ conditioned on $\sigma_1 = i$ is identically distributed to $M_i(x^*)$, so:

$$\begin{aligned}
\textsc{icost}(Q_i, \mathcal{V}^k) &= I(x^* : M_i(x^*) \,|\, X^{(2)}) \\
&= I(x^* : M(X^{(1)}) \,|\, X^{(2)}, \sigma_1 = i) \\
&= I(X_i^{(1)} : M(X^{(1)}) \,|\, X^{(2)}, \sigma_1 = i) \\
&\leq I(X_i^{(1)} : M(X^{(1)}) \,|\, X^{(2)}, \sigma_1) + O\left(\sqrt{\frac{\log n}{n}}\right)
\end{aligned}$$

With the last line following from Lemma 4.2.9.

Now, consider the information cost of P over $\mathcal{U}^k$. We can split up the bits of $X^{(1)}$ using Fact 2.2.17, as they are independent conditioned on σ_1 and $X^{(2)}$. We can also upper bound the mutual information with the entropy of the message, which

is at most the length of the message:

$$\begin{aligned} l_1 \geq H(M(X^{(1)})) \geq \textsc{icost}(P,\mathcal{U}^k) &= I(X^{(1)} : M(X^{(1)}) \mid X^{(2)}, \sigma_1) \\ &\geq \sum_{i\in[n]} I(X_i^{(1)} : M(X^{(1)}) \mid X^{(2)}, \sigma_1) \\ &\geq \sum_{i\in[n]} \left(\textsc{icost}(Q_i, \mathcal{V}^k) - O\left(\sqrt{\frac{\log n}{n}}\right)\right) \end{aligned}$$

Rearrange and divide by n for the result. □

Once we have a protocol for $\textsc{Toy}_k$, we need a way to modify the protocol to remove player 1's message.

Lemma 4.2.16. *Suppose we have a protocol Q where player 1 has a private coin and all other players act deterministically. Suppose Q solves instances of $\textsc{Toy}_n^k$ sampled from $\mathcal{V}^k$ failing with probability ϵ, with fixed message lengths $(l_1, l_2, \cdots, l_{k-1})$. We can create a deterministic protocol Q' for the same problem distribution with message lengths $(0, l_2, \cdots, l_{k-1})$ and failure probability at most $\epsilon + \sqrt{\frac{\ln 2}{2}\textsc{icost}(Q, \mathcal{V}^k) + O\left(\sqrt{\frac{\log n}{n}}\right)}$.*

Proof. Consider inputs to Q. Let A, B, C be random variables: $A = x^*$, $B = X^{(2)}$, and $C = M_Q(x^*)$ - the message sent by player 1 in Q. Let f be a function which takes A, B, C and returns the expected probability that the algorithm will succeed on an input sampled from $\mathcal{V}^k$ given A, B, C. Clearly f maps into $[0, 1]$, and $\mathbb{E}_{A,B,C}[f(A, B, C)] = \epsilon$.
Now, by Lemma 4.2.10, there exists a function g such that:

$$\begin{aligned} \mathbb{E}_{A,B}[f(A, B, g(B))] &\leq \epsilon + \sqrt{\frac{\ln 2}{2} \cdot (I(A : C \mid B) + I(A : B))} \\ &\leq \epsilon + \sqrt{\frac{\ln 2}{2} \cdot \left(\textsc{icost}(Q, \mathcal{V}^k) + O\left(\sqrt{\frac{\log n}{n}}\right)\right)} \end{aligned}$$

where $I(A : B)$ is bounded by Lemma 4.2.7.
So, we can create a protocol Q' where player 1 sends no message, and player 2 acts as if they received $g(B) = g(X^{(2)})$. The failure probability over $\mathcal{V}^k$ is bounded by the above expression and, as the simulated message is deterministic, the protocol is entirely deterministic. □

We now have all the pieces we need to prove the result.

Theorem 4.2.3 (restated)**.** *There is a constant $C > 0$ such that: any communication scheme $\mathcal{B}$ which solves $\textsc{Chain}_n^k$ for $k \leq C\left(\frac{n}{\log n}\right)^{\frac{1}{4}}$ must communicate at least $\Omega\left(\frac{n}{k}\right)$ bits in total.*

Proof. Suppose there is a randomised protocol for arbitrary instances of $\textsc{Chain}_k$, which fails with probability $\epsilon = \frac{1}{3}$. We can assume it has fixed message lengths $(l_1, \cdots, l_{k-1})$, such that $m = \sum_i l_i$. By Yao's principle, this implies that there is a deterministic protocol P for inputs taken from distribution $\mathcal{U}^k$ with the same error chance and message lengths.

Now consider the protocols Q_i we could make from P as in Lemma 4.2.15. We know that their average error is bounded by ϵ, and their average information cost is bounded by $\frac{l_1}{n} + O\left(\sqrt{\frac{\log n}{n}}\right)$. Since, the square root is a concave function, by Jensen's inequality we can find an $i^* \in [n]$ such that:

$$\epsilon_{i^*} + \sqrt{\frac{\ln 2}{2} \cdot \textsc{icost}(Q_{i^*}, \mathcal{U}^{k-1})} \leq \epsilon + \sqrt{\frac{\ln 2}{2} \cdot \left(\frac{l_1}{n} + O\left(\sqrt{\frac{\log n}{n}}\right)\right)} = \epsilon'$$

Taking Q_{i^*} and applying Lemma 4.2.16, we can create a deterministic protocol Q' for $\mathcal{V}^k$ with message lengths $(0, l_2, \cdots, l_{k-1})$ and failure chance at most ϵ'. From Fact 4.2.13, this immediately gives us a protocol for $\mathcal{U}^{k-1}$ with the same properties.

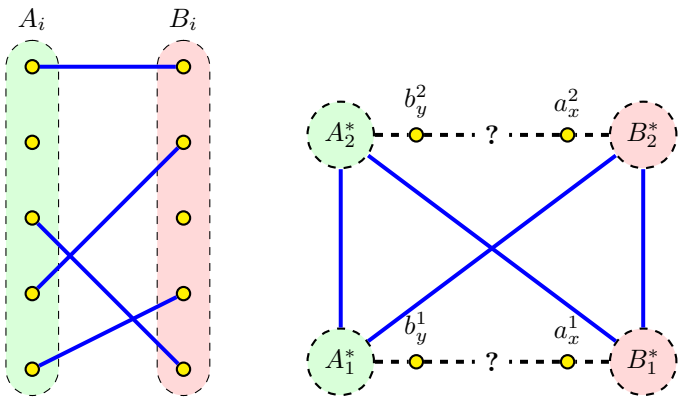
We can now repeat the process of applying Lemmas 4.2.15 and 4.2.16 to iteratively create protocols for $\mathcal{U}^{k-2}$, then $\mathcal{U}^{k-3}, \cdots, \mathcal{U}^2$, with slowly increasing error.

The resulting protocol for $\mathcal{U}^2$ has message lengths (l_{k-1}) and error probability at most

$$\begin{aligned}
&\epsilon + \sum_{i=1}^{k-1} \sqrt{\frac{\ln 2}{2} \cdot \left(\frac{l_i}{n} + O\left(\sqrt{\frac{\log n}{n}}\right)\right)} \\
&\leq \epsilon + \sqrt{\frac{\ln 2}{2} \cdot \left(\frac{\sum_{i=1}^{k-1} k \cdot l_i}{n} + k^2 \cdot O\left(\sqrt{\frac{\log n}{n}}\right)\right)} \\
&\leq \frac{1}{3} + \sqrt{\frac{\ln 2}{2} \cdot \left(\frac{mk}{n} + k^2 \cdot O\left(\sqrt{\frac{\log n}{n}}\right)\right)}
\end{aligned}$$

Then there is a constant C such that if $m \in o\left(\frac{n}{k}\right)$ and $k < C\left(\frac{n}{\log n}\right)^{\frac{1}{4}}$, then for large enough n, this gives a δ-error deterministic protocol for uniformly sampled instances of $\textsc{Index}$ with message length $l_{k-1} \leq o(n)$ and $\delta < \frac{1}{2}$. However this is well known to be impossible, due to Theorem 2.2.10. □

Now that we have all our lower bound tools, we can begin our investigation of the streaming complexity of maximal and maximum independent set in earnest.



(a) Alice encodes X in each G_i.

(b) Bob adds all inter-G_i edges, except those that would touch a_x^1, a_x^2, b_y^1, and b_y^2.

Figure 4.1: Construction of the gadget used to prove Theorem 4.3.1.

4.3 Arbitrary Graphs

In this section we begin our study of the streaming complexity of independent set problems by considering the most general case: when the input graph G has no restriction at all on its structure.

As we discussed in Section 4.1, maximal independent set can be solved in $\tilde{O}(n)$ space in both explicit and implicit vertex arrival streams for any kind of graph. But can it still be solved efficiently in edge streams?

At the same time, we saw that c-approximate maximum independent set has space complexity $\tilde{\Theta}(n^2/c^2)$ in insert-only edge streams of arbitrary graphs. Will this problem remain intractable if we guarantee the nicer structure of a vertex stream?

We will address both of these questions.

4.3.1 Maximal Independent Set in Edge Streams

We begin with our question about the complexity of finding a maximal independent set in an insert-only edge stream. Unfortunately, it turns out that any $\frac{1}{3}$-error single-pass algorithm must use at least $\Omega(n^2)$ bits of space - essentially storing the entire graph.

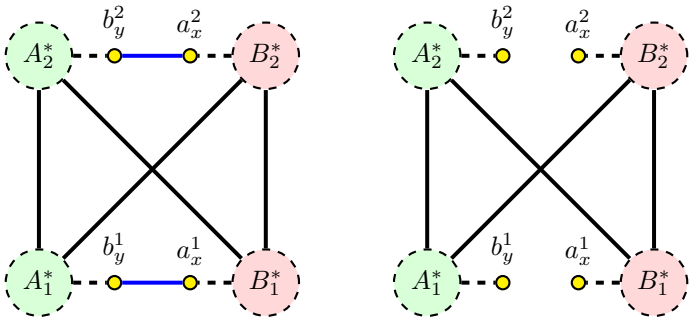
Theorem 4.3.1. *Every $\frac{1}{3}$-error randomised one-pass insert-only edge streaming algorithm that returns a maximal independent set must use at least $\Omega(n^2)$ bits of space.*

Proof. Suppose there is a $\frac{1}{3}$-error randomised one-pass insert-only edge streaming algorithm $\mathcal{A}$ for maximal independent set which uses at most $B(n)$ bits of space for an n-vertex graph.
Consider any even integer n and an instance (X, σ) of the two-party communication problem $\textsc{Index}_N$ with number of bits $N = n^2/4$.

Alice's Edges Given X, we have Alice construct a bipartite graph $G_1 = (A_1, B_1, E_1)$ with vertex sets of size $|A_1| = |B_1| = n/2$. Enumerate these sets as $A_1 = \{a_i^1\}_{i\in[n]}$ and $B_1 = \{b_j^1\}_{j\in[n]}$.
We use the vector X as the adjacency matrix for G_1: so each edge $\{a_i^1, b_j^1\}$ is included in E_1 if and only if $X_{(i-1)n+j} = 1$.
Now create a copy $G_2 = (A_2, B_2, E_2)$ of G_1 and let $G = G_1 \sqcup G_2$ be the disjoint union of graphs G_1 and G_2. The copy of each vertex a_i^1 and b_j^1 is labelled a_i^2 and b_j^2 respectively.
Alice now simulates algorithm $\mathcal{A}$ on some stream ordering of the edges of G and sends the algorithm state $\mathcal{M}$ to Bob. We know this message has bounded size $|\mathcal{M}| \leq B(n)$.
The algorithm guarantees that Bob can perform additional edge insertions (although with no repetitions) and then recover a maximal independent set with probability at least $\frac{2}{3}$.

Bob's Edges Given the algorithm state $\mathcal{M}$ and the index σ, Bob identifies the pair of vertices $(a_x^1, b_y^1) \in A_1 \times B_1$ that corresponds to the index σ. That is, $\sigma = (x-1)n + y$.
Let $A_1^* = A_1 \setminus \{a_x^1\}$, similarly: $A_2^* = A_2 \setminus \{a_x^2\}$, $B_1^* = B_1 \setminus \{b_y^1\}$, and $B_2^* = B_2 \setminus \{b_y^2\}$.
We now need to use Bob's insertions to force every legal maximal independent set to reveal the presence of edge $\{a_x^1, b_y^1\}$.
Bob takes memory state $\mathcal{M}$ and continues algorithm $\mathcal{A}$ by adding all edges between the two sets of vertices $(A_1^* \cup B_1^*)$ and $(A_2^* \cup B_2^*)$. This cannot cause a repetition, because no edges were previously present between G_1 and G_2. This step is represented in Figure 4.1.

Output Now let I be the output maximal independent set computed by the algorithm (conditioned on the algorithm succeeding). We have two cases to consider: either $X_\sigma = 0$ and $X_\sigma = 1$. The cases are illustrated in Figure 4.2.
First, suppose that $X_\sigma = 1$. Then, both the edges $\{a_x^1, b_y^1\}$ and $\{a_x^1, b_y^1\}$ are present in G, so neither can be a subset of I.
On the other hand, consider the case that $X_\sigma = 0$. We will show that at least one pair $\{a_x^1, b_y^1\}$ or $\{a_x^1, b_y^1\}$ is a subset of I.
Suppose that $\{a_x^1, b_y^1\}$ is not fully contained in I. Since I is maximal, that means there is some vertex $z \in A *_1 \cup B_1^*$ contained in I which is adjacent to one of a_x^1



(a) When $X_\sigma = 1$, clearly a_x^i and b_x^i cannot both be in the output.

(b) When $X_\sigma = 0$, any correct output contains $\{a_x^i, b_y^i\}$ for at least one of $i = 1, 2$.

Figure 4.2: Sketch proof for Theorem 4.3.1 that the index bit can be recovered.

or b_y^1.
But then z is adjacent to all of $A_2^* \cup B_2^*$, so none of them can be in I or it would not be an independent set. Therefore, both a_x^2 and b_y^2 must be contained in I because I is maximal and they have no other neighbours.
This means Bob can, with probability at least $\frac{2}{3}$, determine whether $X_\sigma = 1$ by looking to see if either $\{a_x^1, b_y^1\}$ or $\{a_x^1, b_y^1\}$ is contained in the algorithm output.
By Theorem 2.2.10, any such protocol must have communicated a message of at least $\Omega(N) = \Omega(n^2)$ bits, so we have that the algorithm space usage $B(n) = \Omega(n^2)$. □

This resolves the complexity of computing an exactly-maximal independent set in an edge stream, but for some applications perhaps an "approximately maximal" independent set would be good enough.

Definition 4.3.2 (Approximate Maximality)**.** *Let $G = (V, E)$ be an n-vertex graph, and let $I \subseteq V$ be an independent set in G.*
Then I is δ-maximal, if and only if $|I \cup N_G[I]| \geq \delta n$.

A δ-maximal independent set I covers a δ-fraction of the vertices. In other words, removing I and its neighbors $N_G[I]$ from the graph leaves at most $(1 - \delta)n$ vertices.

We will next show that even achieving $\frac{24}{25}$-maximality in insert-only edge streams still requires strictly more space than computing an exactly maximal independent set in a vertex stream.

Further, our lower bound yields that computing a $(1 - \frac{1}{n^\epsilon})$-maximal independent set requires space $\Omega(n^{2-o(1)})$, for every $\epsilon > 0$.

Central to our extended construction are Ruzsa-Szemerédi graphs.

Definition 4.3.3 (Ruzsa-Szemerédi graph)**.** *A bipartite graph $G = (V, E)$ is an (r, s)-**Ruzsa-Szemerédi graph** if its edge set can be partitioned into r induced matchings each of size s.*
In other words, there exists a partitioning $E = \bigcup_{i \in [r]} M_i$, such that each M_i satisfies:

- *The edge set is of size exactly $|M_i| = s$.*
- *All edges of M_i are pairwise disjoint:*

$$e_1, e_2 \in M_i \implies |e_1 \cap e_2| \neq 1$$

- *M_i is the edge set of an induced subgraph of G:*

$$V_i = \bigcup_{e \in M_i} e \implies G[V_i] = (V_i, M_i)$$

Our lower bound for approximate maximality is obtained by a reduction from the two-party communication problem RS-Index, defined as follows:

Definition 4.3.4 (RS-Index$_{r,s}$)**.** RS-Index$_{r,s}$ *is a two-party one-way communication problem set up as follows.*
Let H be an (r, s)-Ruzsa-Szemerédi graph with induced matchings $M_1, M_2, \ldots, M_r$. Both parties know H.
For each induced matching M_i, let $M_i' \subseteq M_i$ be a uniform random subset of size $s/2$ (we assume that s is even). Alice initially holds the graph $G = H[\cup_i M_i']$. Alice may send a single message to Bob.
Bob holds an index $i \in [r]$. Bob receives the message from Alice and then must output $C \cdot s$ edges of M_i', for an arbitrarily small constant C.

Observe that this problem is similar in spirit to Index: In Index, Bob needs to learn one bit, while in RS-Index, Bob needs to learn the presence of many edges of M_i'. A lower bound on the communication complexity of RS-Index is implicit in the work of Goel, Kapralov, and Khanna [GKK12]:

Theorem 4.3.5 ([GKK12])**.** *The randomized constant error communication complexity of* RS-Index *is $\Omega(r \cdot s)$.*

Equipped with the RS-Index problem, we now give a reduction to approximate maximality from RS-Index, which yields our lower bound for streaming algorithms.

Lemma 4.3.6. *Let r, s, n be integers such that there is an n-vertex (r, s)-Ruzsa-Szemerédi graph. Then, every $\frac{1}{3}$-error randomized one-pass insert-only edge streaming algorithm that computes a $(1 - \frac{s}{6n})$-maximal independent set requires $\Omega(r \cdot s)$ bits of space.*

Proof. Let H be an n-vertex (r, s)-Ruzsa-Szemerédi graph, and let G be Alice's input graph for the RS-Index problem derived from H.
Let $M_1, M_2, \ldots, M_r$ denote the induced matchings in H, let $V_i = V(M_i)$, and let $M'_i \subseteq M_i$ denote the subset of edges of matching M_i that is included in G.
Let i be Bob's input. Furthermore, let $\mathcal{A}$ be a $\frac{1}{3}$-error randomized one-pass streaming algorithm for the edge-arrival model that computes a $(1 - \frac{s}{6N})$-maximal independent set on a graph on N vertices. We now show how $\mathcal{A}$ can be used to solve RS-Index:

Alice's Edges Given G, let $\tilde{G}$ be the graph obtained from G, where every induced matching M'_i in G is replaced by edges $\tilde{M'_i} := M_i \setminus M'_i$. Observe that $E(G) \cup E(\tilde{G}) = E(H)$.
Alice now constructs two disjoint copies G_1 and G_2 of $\tilde{G}$, runs algorithm $\mathcal{A}$ on $G_1 \dot{\cup} G_2$ (on an arbitrary ordering of their edges), and sends the memory state to Bob.

Bob's Edges Bob constructs the edge set F that connects every vertex $v_1 \in V(G_1) \setminus V_{i1}$ with every vertex $v_2 \in V(G_2) \setminus V_{i2}$, where V_{i1} and V_{i2} are the copies of the vertices V_i in graphs G_1 and G_2, respectively, and continues the execution of $\mathcal{A}$ on F.

Output Let I be the independent set produced by algorithm $\mathcal{A}$.
Observe that the graph processed by algorithm $\mathcal{A}$ contains $N = 2n$ vertices. Since I is $(1 - \frac{s}{6N})$-maximal, we have $|V \setminus \Gamma[I]| \leq N - (1 - \frac{s}{6N})N = s/6$. This allows us to identify $\Omega(s)$ edges of M'_i as follows:
Let a, b be the incident vertices to an arbitrary edge of M'_i, let a_1, b_1 be the copies of a, b in G_1, and let a_2, b_2 be the copies of a, b in G_2. Observe that a_1 and b_1 are not connected in G_1, and a_2 and b_2 are not connected in G_2.
We now claim that if all vertices a_1, b_1, a_2, b_2 are covered by I, i.e., $\{a_1, b_1, a_2, b_2\} \subseteq \Gamma[I]$, then either $\{a_1, b_1\} \subseteq I$ or $\{a_2, b_2\} \subseteq I$ (or both). Indeed, suppose that this is not the case. Then there are vertices $x_1 \in \{a_1, b_1\}$ and $x_2 \in \{a_2, b_2\}$ with $x_1, x_2 \notin I$. Let $y_1 \in I$ be a vertex incident to x_1, and let $y_2 \in I$ be a vertex incident to x_2. By the construction of the input graph, $y_1 \in V(G_1) \setminus V_{i1}$, and $y_2 \in V(G_2) \setminus V_{i2}$. Observe, however, that the edge $y_1 y_2$ was included by Bob, which implies that y_1, y_2 are not independent: a contradiction.
Hence, either $\{a_1, b_1\} \subseteq I$ or $\{a_2, b_2\} \subseteq I$ (or both) hold. This implies that the

algorithm identified that there is no edge between a_1, b_1, which in turn implies that we learned one edge of M'_i. Hence, for every pair of vertices a, b of M'_i, either at least one vertex among $\{a_1, b_1, a_2, b_2\}$ is not covered by I, or we learn one edge of M'_i.
Since there are $s/2$ edges in M'_i, and at most $s/6$ vertices of the input graph are not covered by I, we learn at least $s/2 - s/6 = \Omega(s)$ edges of M'_i, which thus solves RS-Index. By Theorem 4.3.5, algorithm $\mathcal{A}$ must have used at least $\Omega(r \cdot s)$ bits of space. □

Goel, Kapralov, and Khanna [GKK12] showed that there are n-vertex (r, s)-Ruzsa-Szemerédi graphs with $r = n^{\Theta(\frac{1}{\log \log n})}$ and $s = (\frac{1}{4} - \epsilon)n$ for every $\epsilon > 0$. Alon, Moitra, and Sudakov [AMS12] showed that there are such graphs with $\Theta(n^{2-o(1)})$ edges such that each matching is of size $n^{1-o(1)}$. Combined with Lemma 4.3.6, we obtain:

Theorem 4.3.7. *Every randomised constant error one-pass edge streaming algorithm that computes a $\frac{24}{25}$-maximal independent set requires space $n^{1+\Omega(\frac{1}{\log \log n})}$, and every such algorithm computing a $(1 - \frac{1}{n^\epsilon})$-maximal independent set requires space $\Omega(n^{2-o(1)})$, for every $\epsilon > 0$.*

4.3.2 Maximum Independent Set in Explicit Vertex Streams

In this section we consider the problem of finding approximately maximum independent sets. Recall that this problem is much harder than maximal independent set: Halldórsson, Sun, Szegedy, and Wang [Hal+12] showed that the optimal strategy for edge streams (both turnstile and insert-only, up to logarithmic factors) is to select a random subset of $O(n/c)$ vertices and store the entire induced subgraph requiring $\tilde{\Theta}(n^2/c^2)$ bits of space.

In vertex streams, we can make this algorithm deterministic. Surprisingly, however, we do not know of any better strategy even with randomisation.

Theorem 4.3.8. *There is a deterministic explicit vertex streaming algorithm for finding a c-approximate maximum independent set which uses $O\left((n^2/c^2) \cdot \log n\right)$ bits of space.*

Proof. The strategy is simply to break up the stream into c blocks of n/c consecutive vertices each. The entire induced subgraph on each block can be stored in the space claimed in the theorem, since we have at most n^2/c^2 edges.
At the end of each block, we compute a maximum independent set on the current induced subgraph. This could take exponential time, but can be done in $O(n/c)$ space (for example, by simply enumerating all possible sets and testing independence). We then keep track of the largest independent set seen across all blocks

using an additional $O((n \log n)/c)$ bits of space.
Now consider a particular maximum independent set I of the whole graph G. Since the blocks partition the vertices, at least one block must contain at least $|I|/c$ vertices of I. Therefore, this block must have a large enough independent set within it to act as a c-approximation for the whole graph.
And so we always find a c-approximate maximum independent set to return. □

For explicit vertex streams and small c, we are able to show that no algorithm can do significantly better. Though the question of the exact complexity remains open.

Theorem 4.3.9. *Any $\frac{1}{3}$-error randomised explicit vertex streaming algorithm for c-approximating $\alpha(G)$ must use at least $\Omega\left(\frac{n^2}{c^6}\right)$ bits of space.*

For ease of argument, we will actually prove an equivalent result for the problem of approximating $\omega(G)$ - the size of the largest clique in G - and then argue that these problems are equivalent.

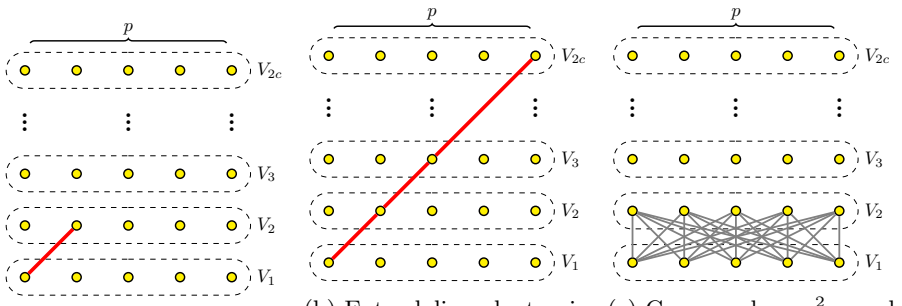
Theorem 4.3.10. *Any $\frac{1}{3}$-error randomised explicit vertex streaming algorithm which finds a c-approximation to the size of the largest clique $\omega(G)$ requires $\Omega\left(\frac{n^2}{c^6}\right)$ bits of space.*

The key to our construction is encoding a length $\Theta(\frac{n^2}{c^4})$ binary vector within an $\Theta(\frac{n}{c})$ vertex induced subgraph, so that each bit corresponds to the presence or absence of a clique of size $2c$. We ensure that no pair of these cliques can share an edge. Then we can chain together $2c$ such gadgets to encode an instance of $\textsc{Chain}^{2c}_{\Theta(n^2/c^4)}$ such that if the correct answer is 1, the resulting graph has an independent set of size $4c^2$, while if the correct answer is 0 the graph has no independent set larger than $4c - 1$. Any c-approximation algorithm could distinguish these two cases, which proves the result.

First we define our clique gadget.

Lemma 4.3.11. *For any positive integers n and $c^2 < \frac{n}{8}$, there exists a graph on n vertices containing $\frac{n^2}{16c^2}$ edge-disjoint cliques of size $2c$ and no cliques of size larger than $2c$.*

Proof. We construct the sets from an erasure code[a] with block size $2c$ and message size 2. Choose a prime p such that $\frac{n}{4c} \leq p \leq \frac{n}{2c}$ (which is guaranteed to exist). Now take $2c < p$ groups of vertices, each of size p. Label the groups V_i (for $i \in [2c]$) and label the items in each group V_i as v^i_j (for $j \in [p]$). Leftover vertices are added to the final graph as isolated vertices.
Let $\mathcal{F}_p$ be the collection of linear functions on the finite field of order p. For each



(a) Select an edge. (b) Extend linearly to size $2c$. (c) Can make p^2 such cliques.

Figure 4.3: Clique gadget construction in Lemma 4.3.11.

polynomial $\mathcal{P} \in \mathcal{F}_p$ we define $K_{\mathcal{P}}$ to be the clique over vertices $\{v^i_{\mathcal{P}(i)} \mid i \in [2c]\}$. This can be viewed as taking each of the p^2 possible edges between V_1 and V_2 and extending them "linearly" to the other layers (see Figure 4.3). In other words, the cliques correspond to non-horizontal lines in the affine plane of order p.

Properties Clearly $\mathcal{K} = \{K_{\mathcal{P}} \mid \mathcal{P} \in \mathcal{F}_p\}$ consists of $p^2 > \frac{n^2}{16c^2}$ cliques, each of size $2c$. We next show that they are pairwise edge-disjoint and that their union contains no larger cliques.

Each clique contains exactly one vertex from each group V_i, so for two cliques to share an edge there must be distinct polynomials $\mathcal{P}, \mathcal{Q} \in \mathcal{F}_p$ that have the same value at two different points: $\mathcal{P}(i) = \mathcal{Q}(i)$ and $\mathcal{P}(j) = \mathcal{Q}(j)$ for $i \neq j$. But this a contradiction to linearity in a prime field.

Finally, because no clique contains a pair of vertices from a single V_i, their union can contain no internal edges on any V_i. So any clique can contain at most 1 vertex from each V_i, giving a maximum size of $2c$.

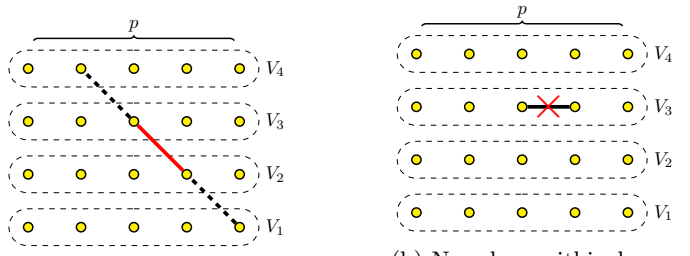
Hence, $\bigcup_{\mathcal{P} \in \mathcal{F}_p} K_{\mathcal{P}}$ is a graph with the required properties. □

[a] In coding theory, an erasure code is a kind of error-correcting code which allows the original message to be recovered from any subset of the code symbols (of at least a certain size). In our case, we encode a 2 symbol message as 2c symbols such that any pair of the code symbols allows us to determine the original message. This allows us to encode a message as a 2c-clique such that knowing any edge is enough to recover the original message - i.e. the cliques are pairwise edge disjoint.

And now we can prove Theorem 4.3.10.

Proof of Theorem 4.3.10. Suppose we have a $\frac{1}{3}$-error explicit vertex streaming algorithm $\mathcal{C}$ which produces a c-approximation to $\omega(G)$, the size of the largest clique. We will show that such an algorithm can be used to solve $\textsc{Chain}^{2c}_N$, where $N = \frac{n^2}{64c^4}$, by communicating its state $2c - 1$ times.

Fix an instance of $\textsc{Chain}^{2c}_N$. Our lower bound in Theorem 4.2.3 implies that any algorithm that can solve this must send at least one message of size $\Omega\left(\frac{N}{c^2}\right) = \Omega\left(\frac{n^2}{c^6}\right)$ bits.



(a) No edge belongs to two different lines (cliques).

(b) No edges within layers, so no cliques larger than $2c$.

Figure 4.4: Clique gadget proof sketch for Theorem 4.3.10.

Take n vertices and partition the nodes into $2c$ groups of size $\frac{n}{2c}$. Each group will be added to the stream by one of the parties.

Within-party edges First, consider the group of nodes associated with party P_i. We will encode the bits of $X^{(i)}$ onto the internal edges of this group using the construction from Lemma 4.3.11. The size $\frac{n}{2c}$ sub-graph can fit N cliques of size $2c$. We include the edges of clique j if and only if $X^{(i)}_j = 1$. This is well defined as the cliques are edge-disjoint. Label the clique in party P_i corresponding to bit j of $X^{(i)}$ as $\mathcal{K}^i_j$. The final party P_{2c} has no associated vector. Instead, it constructs a single clique of size $2c$ and leaves the other vertices isolated.

Between-party edges We also need edges between the sub-graphs associated with different parties. Each party P_i will connect all its vertices to some of the vertices belonging to previous parties (P_j for $j < i$). These edges are considered to belong to party P_i, as they will be added by this party in the vertex streaming model.
For each $j < i$ the party P_i connects every one of its vertices to all of $\mathcal{K}^j_{\sigma_j}$ (the clique corresponding to index σ_j). For this to happen, P_i must know all σ_j for $j < i$. This information is not known initially, but can be appended to the communications between players with only $O(c)$ overhead.

Clique Size Bounds Now that we have our construction, we need to show bounds on $\omega(G)$ for the two cases. First, consider when every $X^{(i)}_{\sigma_i} = 1$. In this case we have each of the cliques $\mathcal{K}^i_{\sigma_i}$ present and connected together, forming a clique of size $4c^2$. Now consider the case when every $X^{(i)}_{\sigma_i} = 0$. Consider a clique $\mathcal{K}$ in the graph. If $\mathcal{K}$ contains multiple vertices belonging to one party P_i, then it can contain none from any subsequent party P_j ($j > i$), and at most one from each preceding party P_l ($l < i$). Hence the size of any clique is bounded by $4c - 1$.
To see why this holds, observe that for any $i < 2c$, our clique can contain only one vertex from $\mathcal{K}^i_{\sigma_i}$, as none of its edges are included in the graph. So to contain

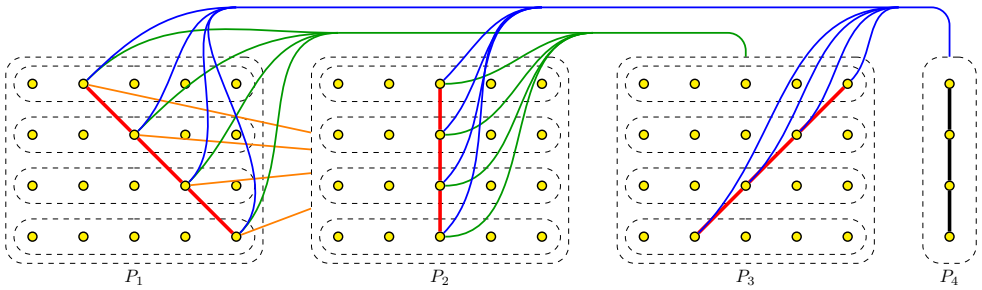


Figure 4.5: Example lower bound instance with 4 players for Theorem 4.3.10. Cliques corresponding to σ_1, σ_2, and σ_3 are shown in bold red—other cliques are omitted.

multiple vertices from party P_i, the clique $\mathcal{K}$ must contain a vertex v from some $\mathcal{K}^i_j$ with $j \neq \sigma_i$. But then all subsequent parties P_j $(j > i)$ will have no vertices adjacent to v, so cannot contribute anything to $\mathcal{K}$. So the best we can do is include one vertex from each $\mathcal{K}^i_{\sigma_i}$ and then $2c$ from party P_{2c} giving a clique of size $4c-1$. To complete the proof, observe that this gap in clique sizes can be distinguished by a c-approximation algorithm, and any streaming algorithm gives a communication protocol by having each party update the algorithm state with their information and then pass it to the next party. □

And then we can return to prove the $\alpha(G)$ version of the theorem.

Proof of Theorem 4.3.9. Repeat the construction, but whenever any party is tasked with inserting i^{th} vertex v_i with edges to each $N_G(v_i) \cap \{v_1, v_2, \cdots, v_{i-1}\}$, instead insert a vertex with complementary edges $\{v_1, v_2, \cdots, v_{i-1}\} \setminus N_G(v_i)$. The resulting graph $\overline{G}$ has $\alpha(\overline{G}) = \omega(G)$ in all cases, and hence it is just as hard to approximate. □

Interestingly, the same construction also gives us hardness for approximating a third quantity: the chromatic number of a graph.

Definition 4.3.12 (Graph Colouring)**.** *Given a graph $G = (V, E)$:*

- *A k-**colouring** of G is a map $f : V \to [k]$ such that for every edge $\{u, v\} \in E$ we have $f(u) \neq f(v)$.*
- *The **chromatic number** $\chi(G)$ of the graph is the smallest integer k such that there exists a legal k-colouring of G.*

Corollary 4.3.13. *Any $\frac{1}{3}$-error randomised explicit vertex streaming algorithm to c-approximate $\chi(G)$ requires $\Omega\left(\frac{n^2}{c^6}\right)$ bits of space.*

Proof. Consider the construction in the proof of Theorem 4.3.9. In the case of all $X^{(i)}_{\sigma_i} = 1$, the graph contains a clique of size $4c^2$, so all these vertices must be

different colours and $\chi(G) \geq 4c^2$.
Conversely, in the case of every $X_{\sigma_i}^{(i)} = 0$, we can construct a $4c$-coloring of the graph.
First color each of the nodes in each $\mathcal{K}_{\sigma_i}^i$ with the i^{th} color (this is allowed, as they have no internal edges). The remaining vertices in each party are then not adjacent to any uncolored vertices from other parties, so we simply need to be able to complete the coloring of each party in isolation with $2c$ new colors and we are finished. This is easily done, as each party's sub-graph is $2c$-partite by construction. □

Notably, the same is not true of the 2-party edge stream construction of Halldórsson, Sun, Szegedy, and Wang [Hal+12]. The random graphs they use as gadgets have large chromatic number with probability $1 - o(1)$ [Bol88]. This means that any chromatic number approximation algorithm would not be able to distinguish between states of the underlying hard communication problem, so no lower bound can be established.

4.3.3 Maximum Independent Set in Implicit Vertex Streams

For the case of implicit vertex streams, the strategy of Theorem 4.3.8 can also be implemented. Storing the at most n/c vertex identifiers for each induced subgraph is enough for us to test set independence, giving the following space cost.

Corollary 4.3.14. *There is a deterministic implicit vertex streaming algorithm for finding a c-approximate maximum independent set which uses $O\left((n/c) \cdot (\log |\mathcal{U}| + \log n)\right)$ bits of space.*

This algorithm is also close to optimal (for small c) among $\alpha(G)$ approximation algorithms which work for any kind of implicit vertex stream.

We prove this by another set disjointness reduction.

Theorem 4.3.15. *Any $\frac{1}{3}$-error randomised implicit vertex streaming algorithm for c-approximating $\alpha(G)$ must use at least $\Omega\left(\frac{n}{c^2} \cdot \frac{\log |\mathcal{U}|}{\log n}\right)$ bits of space for $\log |\mathcal{U}| \geq \log n$.*

Proof. This result arises because the multi-party set disjointness problem can be almost directly encoded as an implicit vertex stream for any desired vertex identifier size $\log |\mathcal{U}|$.

Adjacency Function Choose an integer $N > 0$ and an integer identifier size s such that $\lceil \log N \rceil \leq s \leq N$. Let $r = \lceil s / \log N \rceil$ and $\mathcal{U} = [N+1]^r$, so we have that $\log |\mathcal{U}| = \Theta(s)$.

Now define the adjacency function $f : \mathcal{U}^2 \to \{0, 1\}$ by:

$$f(x, y) = \begin{cases} 0 \text{ if } x_i = y_j \leq N \text{ for some } i, j \in [r] \\ 1 \text{ otherwise} \end{cases}$$

So each identifier encodes r indices from $[N]$, and a group of vertices are independent only when they all share at least one of these indices in common. The spare index $N + 1$ is used for padding.

Stream Simulation Now, suppose we have a $\frac{1}{3}$-error randomised implicit streaming algorithm $\mathcal{A}$ for c-approximating $\alpha(G)$, which uses at most $B(n)$ bits of space for an n-vertex graph. Choose integer $c \geq 1$ and pick an instance of the $2c$-party communication problem DISJOINT_N^{2c}.

The first party P_1 knows X_1. They begin simulating algorithm $\mathcal{A}$. We wish to represent the set X_1 as $\lceil |X_1|/r \rceil$ vertices of the graph. Group the members of the set arbitrarily into $\lceil |X_1|/r \rceil$ multisets of size r, padding one of the multisets with copies of index $N + 1$ to make this possible. Now for each group, we simulate the insertion of the identifier consisting exactly of the indices in that group (arranged arbitrarily). Then pass the algorithm state $\mathcal{M}$ to party P_2.

Now, for each party P_k for $k \leq 2c$, have P_k continue simulating algorithm $\mathcal{A}$. They should also group X_k arbitrarily into multisets of size r and insert corresponding vertex identifiers into the stream, in the same manner as party P_1. Then, if $k < 2c$, pass the state $\mathcal{M}$ to the next party.

Party P_{2c} should perform some additional dummy insertions of $\lceil N/r \rceil$ copies of $(N + 1)_{i \in [r]}$, to ensure we have roughly the right graph size.

Output Once every party has gone, let I be the combined multiset of inserted identifiers and let G_I be the graph of I (according to adjacency function f).

If $\bigcap_{k \in [2c]} X_k = \emptyset$, then (by the definition of the set disjointness problem) each $i \in [N]$ can only be in a unique X_k. This means that the total size $\sum_{k \in [2c]} |X_k| \leq N$. We bundled the elements into groups of size r, so we added identifiers for at most $\lceil n/r \rceil + 2c$ groups in total (the extra $2c$ coming from having up to one extra group per party due to rounding). Then the final party added a further $\lceil N/r \rceil$ vertices, for a total of: $\lceil N/r \rceil \leq |I| \leq 2\lceil N/r \rceil + 2c$.

So we have that the number of vertices $n = \Theta(N/r + c)$. Further, no index other than $N+1$ was included in multiple identifiers, so the entire graph is a clique with $\alpha(G_I) = 1$.

On the other hand, if $\bigcap_{k \in [2c]} X_k = \{j\}$, then each $i \in [n] \setminus \{j\}$ is still only included in a unique X_k but the special index j is in every X_k. So, this time we have $\sum_{k \in [2c]} |X_k| \leq N + 2c$. Then, by a similar argument: $\lceil (N + 2c)/r \rceil \leq |I| < \lceil N/r \rceil + \lceil (N + 2c)/r \rceil + 2c$.

So we have that the number of vertices is still $n = \Theta(N/r+c)$. Further, exactly one

identifier per party contained the index j while no other indices (other than $N+1$) were repeated, so the independence number of the graph is exactly $\alpha(G_I) = 2c$. This gap is large enough for the c-approximation algorithm to distinguish, so by Theorem 2.2.12 we must have communicated at least one message of size $\Omega(N/c^2)$. Therefore $B(n) = \Omega(nr/c^2) = \Omega(n \log |\mathcal{U}|/c^2 \log n)$, as required. □

4.4 Locally-Bounded Independence Graphs

As we saw in the previous section, we cannot hope to find good approximations to $\alpha(G)$ in arbitrary graphs, even for vertex streams, unless we essentially store the entire stream. However, this does not mean we cannot do better for special classes of graphs.

In this section, we consider vertex streams over restricted families of graphs that have *polynomially-bounded independence*.

Definition 4.4.1 (Bounded Independence)**.** *Given a function $f : \mathbb{N} \to \mathbb{N}$, we say a family of graphs $\mathcal{F}$ has **f-bounded independence** if:*

For every vertex v in every graph $G \in \mathcal{F}$, and for every integer $r \geq 1$, we have that the r-neighbourhood[a] N of v satisfies $\alpha(G[N]) \leq f(r)$.

That is, no r-neighbourhoods in any graph of the family contains an independent set of size larger than $f(r)$.

*When f is a polynomial, we say the family has **polynomially-bounded independence**.*

[a] The set of vertices connected to v by a path of length at most r.

Examples of graph classes that have polynomially-bounded independence include unit interval intersection graphs (with $f(r) = 2r + 1$) and unit disk intersection graphs (with $f(r) = (2r + 1)^2$) [HK15, Lemma 3].

Consider any family of graphs $\mathcal{F}$ with f-bounded independence. For such graphs, any maximal independent set is also an $f(1)$-approximate maximum independent set, since each vertex in the maximal set can only block at most $f(1)$ other vertices from being included. This means that the GreedyIS algorithm gives us an efficient $\tilde{\mathrm{O}}(\alpha(G)) \leq \tilde{\mathrm{O}}(n)$ space deterministic algorithm for $f(1)$-approximating maximum independent in vertex streams restricted to the graph family $\mathcal{F}$.

This is space-optimal for any approximate maximum independent set algorithm, as we need at least enough space to store the output which can be as big as $\alpha(G)$ words. However, we can hope to do better if the task is simply to approximate the value of the quantity $\alpha(G)$ rather than finding an actual independent set.

Can we produce polylog space algorithms for approximating $\alpha(G)$ for families of bounded independence graphs over vertex streams? In this section we will address exactly this question.

4.4.1 The Caro-Wei Bound Quantity

Recalling that $\alpha(G)$ is the maximum independent set size in G, we introduce a new quantity $\beta(G)$ based on the graph degree distribution.

Definition 4.4.2. *For a graph $G = (V, E)$ the **first negative frequency moment** is given by:*

$$\beta(G) = \sum_{v \in V} \frac{1}{deg(v) + 1}$$

This quantity is known to be interesting in the study of the size of maximum independent sets. Caro and Wei [Car79; Wei81] showed (independently) that every graph contains an independent set of at least this size.

Fact 4.4.3. *For any graph G it must be that $\beta(G) \leq \alpha(G)$.*

In general $\beta(G)$ can be very poor as an approximation to $\alpha(G)$. For example, a dense bipartite graph can have small constant $\beta(G) = n/(1 + n/2) \leq 2$ despite containing an independent set of linear size $\alpha(G) = n/2$. However, it has been shown that the discrepancy cannot be too big for graphs with polynomially-bounded independence.

Theorem 4.4.4 (Theorem 1 in [HK15])**.** *Suppose the family of graphs $\mathcal{F}$ has f-bounded independence for some polynomial f.*
Then there is a constant $C > 0$, so that for any $G \in \mathcal{F}$ we have:

$$\alpha(G) \leq C \cdot \beta(G) \cdot f\left(\frac{\log n}{\log \log n}\right)$$

Recall that the family of unit disk intersection graphs has f-bounded independence with $f(n) = (2n + 1)^2$, so $\beta(G)$ can act as an $O(\log^2 n / \log \log^2 n)$-approximation to $\alpha(G)$ for any unit disk intersection graph G. This is worse than the $f(1)$-approximation achieved by GreedyIS, but we can hope to be able to estimate $\beta(G)$ in sublinear space.

4.4.2 Tight Bounds for Approximation in Streams

Our first set of results in this section are tight bounds on the problem of estimating $\beta(G)$ in insert-only vertex or edge streams.

Braverman and Chestnut [BC15] studied the problem of estimating negative frequency moments (and similar quantities) of vectors given by insert-only streams. But graph degree distributions have slightly more structure than a vertex stream. As such, any upper bounds for vertex streams will apply to graph streams, but the lower bounds do not necessarily carry across.

First, we show for reference a simple sampling upper bound that can be implemented in turnstile edge streams (and therefore in insert-only edge streams and explicit vertex streams).

Theorem 4.4.5. *There is a $\frac{1}{3}$-error randomised p-pass turnstile edge streaming algorithm which returns a c-approximation to $\beta(G)$ for $\beta(G) \geq \gamma$ using $\mathrm{O}(n \log^3 n/p\gamma c)$ bits of space. Where G is a simple graph this can be reduced to $\mathrm{O}(n \log n/p\gamma c)$ bits.*

Proof. Let $r = 3/\gamma(\sqrt{c}-1)^2$. We assume $r/p < 1$, otherwise we may as well store the degree of every vertex and calculate $\beta(G)$ exactly.

Estimator Now consider the following simple estimator - uniformly at random select p subsets of the vertices $U_i \subset V$ for $i \in [p]$, each of size $|U_i| = nr/p$, then let:

$$\beta'_i = \frac{p}{r} \sum_{v \in U_i} \frac{1}{\deg_G(v)+1} \text{ for each } i \in [p]$$
$$\beta' = \frac{1}{p} \cdot \sum_{i \in [p]} \beta'_i$$

Clearly, by linearity of expectation, we have that each $\mathbb{E}[\beta'_i] = \beta(G)$, since each vertex v contributes $p/r(\deg_G(v)+1)$ to the sum with probability r/p and otherwise 0. This means that $\mathbb{E}[\beta'] = \beta(G)$ as well.
Now consider the variance $\mathbb{V}[\beta']$. Each β'_i is independent, so we immediately have that $\mathbb{V}[\beta'] = \frac{1}{p^2}\sum_{i\in[p]} \beta'_i$ by the basic properties of variance.
So let us look at $\mathbb{V}[\beta'_i]$ for some $i \in [p]$. We are essentially sampling without replacement, so the covariance between the contributions from different vertices to β'_i is negative. Therefore, we can bound the variance by the sum of the variances of the vertex contributions, which are simply scaled Bernoulli random variables:

$$\begin{aligned}\mathbb{V}[\beta'_i] &\leq \sum_{v \in V} \frac{r}{p}\left(1-\frac{r}{p}\right)\left(\frac{p}{r(\deg_G(v)+1)}\right)^2 \\ &\leq \sum_{v \in V} \frac{p}{r(\deg_G(v)+1)^2} \leq \frac{p}{r} \cdot \beta(G)\end{aligned}$$

So then:

$$\mathbb{V}[\beta'] = \frac{1}{p^2}\sum_{i\in[p]} \beta'_i \leq \frac{1}{r}\beta(G)$$

Now recall Chebychev's inequality and apply it to $\beta'(G)$:

$$\mathbb{P}\left[|\beta' - \beta(G)| \geq (\sqrt{c}-1)\beta(G)\right] \leq \frac{\mathbb{V}\left[\beta'(G)\right]}{(\sqrt{c}-1)^2\beta(G)^2} \leq \frac{1}{r(\sqrt{c}-1)^2\beta(G)} \leq \frac{1}{3}$$

So with probability at least $\frac{2}{3}$, we have that $\beta'(G)$ is a c-approximation to $\beta(G)$. To complete the proof, we need to describe how this estimator can be calculated in each kind of stream.

Simple Streams First consider a p-pass turnstile edge stream describing a simple graph. At the start of the i^{th} pass randomly choose the subset U_i and record it using $O(n \log n/p\gamma c)$ bits of space. Now we simply keep a count of the degree of each of these vertices, using the same space again. Then at the end of each pass we can calculate β_i' and add it to a running sum which we can assume uses only $O(\log n)$ bits of precision. By the end of the stream, we have the sum $\sum_{i\in[p]} \beta_i'$, which we can rescale to get our desired output.

Multi-Graph Streams Now consider a p-pass turnstile edge stream describing a multi-graph. We wish to use broadly the same strategy, but we need to avoid counting repetitions of the same edges. This can be achieved using an l_0 estimation sketch, also called a count-distinct sketch, using $O(\log^3 n)$ bits of space per vertex. □

The same strategy does not work exactly for implicit edge streams, since we cannot count exactly the degrees in a sampled subset of the vertices. For any vertex sampled late in the stream, we do not know how many earlier vertices it was adjacent to.

As we said earlier, while it is clear that any vector algorithm can be implemented as an edge streaming algorithm, it is not automatically true that the vector lower bounds apply. It could be the case that the additional structure imposed on graphs - that vertex degrees can only be increased by also increasing the degree of another vertex, and only once per pair - could give us more algorithmic power; especially in the case of explicit vertex streams, where there is even more structure.

Unfortunately, our next result shows that the additional structure is not enough to help us, even for explicit vertex streams. We adapt ideas from the vector lower bounds of Braverman and Chestnut [BC15] to build a hard family of graphs for our problem and show that they can be constructed in a way which form valid explicit vertex streams.

Theorem 4.4.6. *Suppose we are promised that $\beta(G) \geq \gamma$ for some integer $\gamma \geq 1$. Then any $\frac{1}{3}$-error randomised p-pass explicit vertex (or insert-only edge) streaming algorithm for c-approximating $\beta(G)$ must use $\Omega(n/p\gamma c)$.*
In particular, for single-pass algorithms, with no promised lower-bound on $\beta(G)$, we must use at least $\Omega(n/c)$ bits.

Proof. Any insert-only edge streaming algorithm is also an explicit vertex streaming algorithm, so we only need to lower bound the latter.
Suppose we have a $\frac{1}{3}$-error randomised p-pass explicit vertex streaming algorithm $\mathcal{A}$ which returns a c-approximation to $\beta(G)$ using at most $B(n)$ bits of space for a simple n-vertex graph G.
We will describe how $\mathcal{A}$ can be used to solve DISJOINT_k with $k = \Theta(n/\gamma c^2)$ by communicating at most $B(n)$ bits. Consider an instance (X, Y) of DISJOINT_k.

Alice's Vertices Given X, Alice can construct a graph $G_1 = (V, E_1)$ as follows. Partition the vertices into disjoint subsets A, B, C and U_i for each $i \in [k]$. These should be of sizes $|A| = |B| = 2k\gamma c$, $|C| = \gamma$, and each $|U_i| = 2\gamma c$. Hence the total number of vertices is $n = |V| = (6kc + 1)\gamma$, meaning that $k = \Theta(n/\gamma c)$.
Consider the two sets of edges:

$$E_A = \bigcup_{u,v \in A} \{\{u, v\}\} \quad \text{and} \quad E_X = \bigcup_{i \in [k] \setminus X} \left(\bigcup_{u \in U_i,\, v \in A} \{\{u, v\}\} \right)$$

E_A makes A into a clique, while E_X connects all the vertices associated with each U_i with $i \notin X$ to all the vertices of A. Let $E_1 = E_A \cup E_X$. Figure 4.6a demonstrates the resulting graph G_1 for $k = 5$ and $X = \{2, 4\}$.
Now Alice runs algorithm $\mathcal{A}$ on the graph G_1. She can have the vertices of A, C, and each U_i arrive in any order, since we will not need to add further edges between them. But she leaves the vertices in B "unarrived", with their incident edges to be defined by Bob. She sends the algorithm state $\mathcal{M}$ to Bob.

Bob's Vertices Now, Bob knows the algorithm state $\mathcal{M}$ and the set Y. Bob wishes to add further edges to graph. Define the two edge sets:

$$E_B = \bigcup_{b \in B,\, x \in (A \cup B)} \{\{b, x\}\} \quad \text{and} \quad E_Y = \bigcup_{i \in [k] \setminus Y} \left(\bigcup_{u \in U_i,\, v \in B} \{\{u, v\}\} \right)$$

E_B makes $A \cup B$ into a clique, while E_Y connects every vertex of each U_i with $i \notin Y$ to all the vertices of B. The combined set of edges $E_2 = E_B \cup E_Y$ is entirely incident upon B, so Bob can add these edges to the graph by continuing to run algorithm $\mathcal{A}$ and inserting the vertices of B in any order.

If algorithm $\mathcal{A}$ is a multi-pass algorithm, we have Bob and Alice continue messaging the algorithm state back and forth performing the same vertex insertions each time, until p passes have been simulated.
Call the final graph $G = (V, E_1 \cup E_2)$. Figure 4.6b demonstrates G for $k = 5$, $X = \{2, 4\}$, and $Y = \{1, 2, 3\}$; while Figure 4.6c shows the case when $k = 5$, $X = \{2, 4\}$, and $Y = \{1, 3\}$.

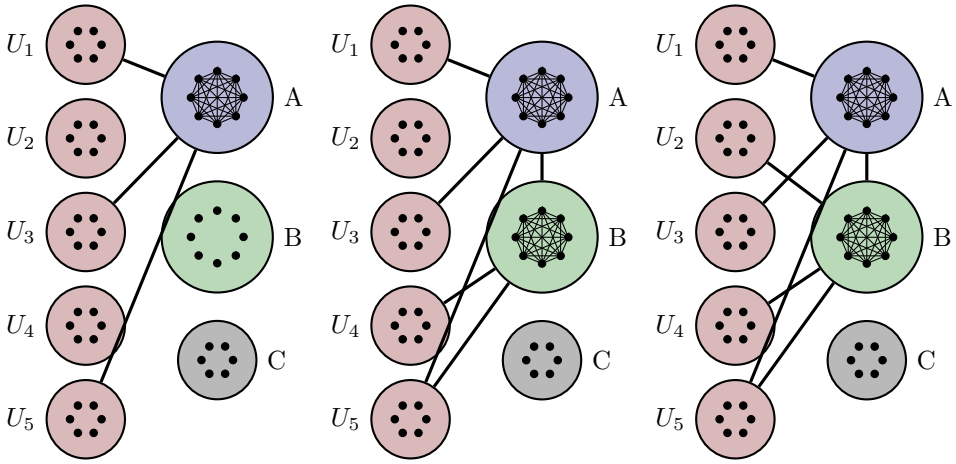
Output Now, consider $\beta(G)$. In the case where $X \cap Y = \emptyset$, we will have edges connecting all of each U_i to all of A or B (or both). This means that the degree of each vertex in each U_i is at least $2k\gamma c$. As there are $2k\gamma c$ vertices in the union of all U_i's, they contribute at most 1 to $\beta(G)$.
Similarly, $A \cup B$ is a clique, so each vertex has degree at least $4k\gamma c - 1$. There are $4k\gamma c$ such vertices, so they also contribute at most 1 to $\beta(G)$.
Vertices in C are isolated and contribute exactly γ to $\beta(G)$. Therefore, $\gamma \leq \beta(G) \leq \gamma + 2$.
Now consider the case where $X \cap Y \neq \emptyset$. This means that there exists some $i \in X \cap Y$, and so all vertices in U_i will have no edges. Since each vertex in $U_i \cup C$ is isolated, and contributes exactly 1 to β, we have that $\beta(G) \geq (2c + 1)\gamma$.
Since our algorithm $\mathcal{A}$ is only wrong by a factor of c with constant probability, we see that the algorithm will allow us to distinguish the two cases (with constant probability), by comparing against the threshold $\tau = (\gamma + 2)c$. If the returned estimate is at most τ, we know that $X \cap Y = \emptyset$ with probability at least $\frac{2}{3}$; otherwise, we know that $X \cap Y \neq \emptyset$ again with probability $\frac{2}{3}$.
From Theorem 2.2.12, we know that Alice and Bob must have communicated at least $\Omega(k/p) = \Omega(n/p\gamma c)$ bits. □

This shows that the simple sampling strategy of Theorem 4.4.5 is optimal up to a $(\log n)$ factor for edge streams (both insert-only and turnstile) and explicit vertex streams.

4.4.3 Bypassing the Lower Bound by Relaxing the Range

Recall our lower bound construction from Theorem 4.4.6. Notice that while $\beta(G)$ is very different between the cases $X \cap Y = \emptyset$ and $X \cap Y \neq \emptyset$, the independence number $\alpha(G)$ is always at least $n/3$.

Since we were only interested in $\beta(G)$ as an approximation to $\alpha(G)$, this suggests a way forwards: if we only ask for an estimator between $\beta(G)/c_1$ and $c_2 \cdot \alpha(G)$, for some $c_1, c_2 > 0$ with $c_1 \cdot c_2 = c$, then Theorem 4.4.6 does not apply and we can hope to find an efficient algorithm which will still be useful for polynomially-bounded independence graphs. So we want an algorithm which returns a c-approximation to $\alpha(G)$ or $\beta(G)$ or any value in between.



(a) G_1 with with $X = \{2, 4\}$. (b) G with $X = \{2, 4\}$ and $Y = \{1, 2, 3\}$. (c) G with $X = \{2, 4\}$ and $Y = \{1, 3\}$.

Figure 4.6: Lower bound construction for Theorem 4.4.6.

To see how we might achieve a better algorithm, consider the following fact.

Fact 4.4.7. *Suppose we have graph $G = (V, E)$ on $n = |V|$ vertices, described by an explicit vertex stream. Let $G_1, G_2, \cdots, G_n$ be the sequence of increasing induced subgraphs where G_i is the subgraph of G induced by the first i vertices to arrive in the stream.*
Then:

$$\beta(G) \leq \max_{i \in [n]} \{\beta(G_i)\} \leq \max_{i \in [n]} \{\alpha(G_i)\} = \alpha(G)$$

Proof. The first inequality holds because $\beta(G) = \beta(G_n)$. The second inequality holds because each $\beta(G_i) \leq \alpha(G_i)$. Then the final equality holds because each G_i is an induced subgraph of G. □

We will show that the quantity $\max_{i \in [n]} \{\beta(G_i)\}$ can estimated efficiently in explicit vertex streams, which is then itself sandwiched between the quantities of interest.

Consider a particular $\beta(G_i)$. We will show how it could be approximated using a small number of vertex counts.

Definition 4.4.8 (Degree Threshold Count)**.** *Consider a graph $G = (V, E)$ with $n = |V|$ vertices.*
For every degree threshold $d \in [n]$, let the number of vertices in G with degree at most d be given by:

$$\mu_d(G) = |\{v \in V \mid \deg_G(v) \leq d\}|$$

Now observe that to get an $O(\log n)$-approximation to any $\beta(G)$ it is enough to estimate the maximum $\mu_{2^i}(G)/(2^i+1)$ over powers $i \in [L]$ for $L = \lceil \log n \rceil$.

Lemma 4.4.9. *Let $L = \lceil \log n \rceil$. For any graph $G = (V, E)$ with $|V| = n$ vertices, we have that:*

$$\frac{\beta(G)}{\log n} \leq \max_{i \in [L]} \left\{ \frac{\mu_{2^i}(G)}{2^i + 1} \right\} \leq 2\beta(G)$$

Proof. For each $i \in [L]$, let $V_i = \{v \in V \mid 2^{i-1} \leq \deg_G(v) < 2^i\}$. Then:

$$\begin{aligned}
\beta(G) = \sum_{v \in V} \frac{1}{\deg_G(v) + 1} &\leq \sum_{i \in [L]} \sum_{v \in V_i} \frac{1}{2^i + 1} \\
&= \sum_{i \in [L]} \frac{\mu_{2^i}(G)}{2^i + 1} \\
&\leq |L| \cdot \max_{i \in [L]} \left\{ \frac{\mu_{2^i}(G)}{2^i + 1} \right\}
\end{aligned}$$

□

Combining Fact 4.4.7 and Lemma 4.4.9, we see the following.

Lemma 4.4.10. *Suppose we have graph $G = (V, E)$ on $n = |V|$ vertices, described by an explicit vertex stream. Let $G_1, G_2, \cdots, G_n$ be the sequence of increasing induced subgraphs where G_i is the subgraph of G induced by the first i vertices to arrive in the stream.*
Then:

$$\frac{\beta(G)}{\log n} \leq \max_{i \in [n]} \left\{ \max_{j \in [L]} \left\{ \frac{\mu_{2^j}(G_i)}{2^j + 1} \right\} \right\} \leq 2\alpha(G)$$

where $L = \lceil \log n \rceil$.

To approximate this quantity, we will estimate $c_j \approx \max_{i \in [n]} \{\mu_{2^j}(G_i)\}$ separately for each $j \in [L]$ and then the calculate $\max_{j \in [L]} \{c_j/(2^j + 1)\}$.

This is possible, because $\max_{i \in [n]} \{\mu_{2^j}(G_i)\}$ can be 2-approximated in explicit vertex arrival streams using a sampling strategy.

Theorem 4.4.11. *For each $d \in [n]$, there is a one-pass $1/n^2$-error randomised explicit vertex streaming algorithm which returns a 2-approximation to $\max_{i \in [n]} \{\mu_d(G_i)\}$ using $O(\log^2 n)$ bits of space, where each G_i is the graph induced by the first i vertices to arrive.*

Proof. Consider Algorithm 2 applied to our explicit vertex stream.
Let $M = \max_{i \in [n]} \{\mu_d(G_i)\}$, the quantity we wish to approximate. If $M \leq \tau$ (where $\tau = 24 \log n$), then at some point we store all the relevant vertices in S and

Algorithm 2: Estimate

```
def Initialise(n, d):
    τ ← 24 log n
    γ ← ∛2
    S ← ∅
    μ ← 0
    p ← 1
def Insert(v′, E′):
    for {u, v′} ∈ E′ do
        if u ∈ S then
            d_u ← d_u + 1
            if d_u > d then
                S ← S \ {u}
                delete d_u
    with probability p do
        if |E′| ≤ d then
            S ← S ∪ {v}
            d_v ← |E′|
            if |S| ≥ τ then
                Remove each vertex from S with probability 1/2
                p ← p/γ
                μ ← τ/p
def Output():
    if μ = 0 then
        return |S|
    else
        return μ
```

return the exact count. For $M > \tau$, we must consider the value of p at the end of the stream.

Recall that $\gamma = \sqrt[3]{2}$. Let r be the unique integer such that $M/\gamma^r < \tau \leq M/\gamma^{r-1}$. We will argue that, at the end of the stream, $p \in \{1/\gamma^{r-1}, 1/\gamma^r, 1/\gamma^{r+1}\}$ with probability at least $(1 - 1/n^2)$. When this holds, the return value $\mu = \tau \cdot \gamma^{r-1}$, $\tau \cdot \gamma^r$, or $\tau \cdot \gamma^{r+1}$ satisfies

$$\gamma^2 \cdot M \geq \mu > M/\gamma$$

which is a 2-approximation to M, since $\gamma^3 = 2$.

Suppose instead that $p \leq 1/\gamma^{r+2}$ at the end of the stream. This means that, at some point in the stream we had $|S| \geq \tau$ and $p = 1/\gamma^{r+1}$. However, at any point in the stream, there are at most M different vertices that can legally be included in S (since we maintain the property that vertices in S currently have degree at most d), each of which is independently included with probability p. Therefore we can bound the distribution of the size of $|S|$ (at any one point in the stream) from above by a binomial distribution with mean $M \cdot p = M/\gamma^{r+1} < \tau/2$. Recalling the tail bounds of Corollary 2.2.4, we know $|S| \geq \tau$ with probability at most $\exp(-\tau/6) = n^{-4}$. We perform at most n updates over the course of the stream, so the chance that this happens at some point is at most n^{-3}.

Now suppose that $p \geq 1/\gamma^{r-2}$ at the end of the stream. Consider the subgraph G_{i^*} that attains the maximum value M of $\{\mu_d(G_i)\}$. As the algorithm finishes processing the update corresponding to vertex i^*, we know that there are exactly M different vertices with degree at most d in the current stream subgraph G_{i^*}, each included in S independently with probability p. So we can bound the distribution of the size of $|S|$ (at this point in the stream) from below by a binomial distribution with mean $M \cdot p \geq M/\gamma^{r-2} \geq 2\tau$. This means that probability that $|S| < \tau$ after that update is processed is at most $\exp(-2\tau/8) = n^{-6}$.

So, at the end of the stream, we have $p \in \{1/\gamma^{r-1}, 1/\gamma^r, 1/\gamma^{r+1}\}$ with probability at least $(1 - 1/n^{24} - 1/n^3) \geq (1 - 1/n^2)$. □

Putting it together, we get our efficient explicit vertex streaming algorithm.

Theorem 4.4.12. *There is a one-pass $1/n^2$-error randomised explicit vertex streaming algorithm which returns a value between $\beta(G)/2 \log n$ and $4\alpha(G)$ using $\log^3 n$ bits of space.*

Proof. Let $L = \lceil \log n \rceil$. For each $j \in [L]$ run a copy of Algorithm 2 with $d = 2^j$ and call the output c_j. This requires $O(\log^3 n)$ bits of space. By Theorem 4.4.11, each c_j is a 2-approximation to $\max_{i \in [n]}\{\mu_{2^j}(G_i)\}$ with probability at least $(1 - 1/n^3)$. Therefore, all L of them are 2-approximations with probability at least $(1 - 1/n^2)$. In this case, Lemma 4.4.10 tell us that:

$$\max_{j \in [L]} \left\{ \frac{c_j}{2^j + 1} \right\}$$

will lie in the claimed range. □

Having found an improved strategy for vertex streams which bypasses our previous lower bound, we may ask whether a similar result is possible over edge streams. Unfortunately, we will find that the answer is no. Even a "relaxed" estimator will require $\tilde{\Omega}(n/c)$ bits of storage for any insert-only edge streaming algorithm.

To prove this lower bound, we will need to be able to construct graphs which contain no large independent sets in themselves or their complements. The following lemma guarantees the existence of such graphs.

Lemma 4.4.13. *There exists a constant $L \geq 1$, such that for any N we can find a graph G on N vertices such that G has small independence number $\alpha(G) \leq L \log N$ and so does its complement $\alpha(\overline{G}) \leq L \log N$.*

Proof. Grimmett and McDiarmid [GM75] showed that as $N \to \infty$, the largest clique in a random graph (where each edge is included with probability $1/2$) is at most $2 \log N + o(\log N)$ with probability $p \to 1$. Therefore, for all large enough N, there must be graphs that satisfy this bound for both themselves and their complements. □

The idea of our lower bound is to associate groups of vertices with each bit of an instance of the INDEX communication problem. We use our above construction to associate half the internal edges of each group with Alice and the other half with Bob. In this way, they can guarantee not introducing any duplicate edges, but either party has the power to prevent a particular group from having a large independent set. Each party uses this power to encode their respective information, such that the resulting graph contains a large $\alpha(G)$ and $\beta(G)$ if and only if the desired INDEX bit was 1.

Theorem 4.4.14. *Choose $c_1, c_2, c \geq 1$ such that $c_1 \cdot c_2 = c$. Let $\gamma \geq 1$.*
Then any single-pass $\frac{1}{3}$-error randomised insert-only edge streaming algorithm which returns an estimate between $\beta(G)/c_1$ and $c_2 \cdot \alpha(G)$ as long as $\beta(G) \geq \gamma$, must use at least $\Omega(n/\gamma c \log c)$ bits of space.
In particular, for streams with no promised lower bound, any such algorithm must use at least $\tilde{\Omega}(n/c)$ bits.

Proof. Suppose that there is a $\frac{1}{3}$-error randomised algorithm $\mathcal{A}$ which returns an estimate between $\beta(G)/c_1$ and $c_2 \cdot \alpha(G)$ and uses at most $B(n)$ bits of space for an n-vertex graph. For convenience let $\lambda = 6L(2 + \log(6Lc))$, where L is the constant from Lemma 4.4.13.

Alice's Edges Consider an instance (X, σ) of the two-party communication problem INDEX$_k$. Given X we will have Alice construct an initially empty graph

$G_1 = (V, \emptyset)$. Partition the vertices into disjoint sets C and U_i for each $i \in [k]$. These sets should be of sizes $|C| = \gamma$ and each $|U_i| = \gamma c\lambda$. So the total number of vertices is $n = |V| = (k\gamma c\lambda + 1)\gamma$, meaning that $k = \Theta(n/\gamma c \log c)$.
We can assume Alice has some deterministic procedure for finding a graph H on $N = \gamma c\lambda$ vertices which satisfies Lemma 4.4.13. For example, enumerating graphs of this size and testing if they meet the requirements.
Now simulate algorithm $\mathcal{A}$ on G_1. For each $i \in [k]$, if $X_i = 0$ she inserts the edges that would be needed to make the induced subgraph $G_1[U_i]$ isomorphic to H (using some deterministic procedure to decide which vertex should be mapped to which). Call the collection of all the inserted edges E_1. Once complete, Alice passes the algorithm state $\mathcal{M}$ to Bob.

Bob's Edges Now Bob knows σ and $\mathcal{M}$. Bob continues simulating algorithm $\mathcal{A}$ to insert two further groups of edges. Define the first edge set by:

$$E_\sigma = \bigcup_{i \neq j \in [k] \setminus \{\sigma\}} \left(\bigcup_{u \in U_i,\, v \in U_j} \{\{u, v\}\} \right)$$

This consists of all edges between different vertex partitions U_i and U_j but excluding the partition U_σ.
Now let $G_2 = (V, E_2)$ where E_2 consists exactly of all the edges to make each induced subgraph $G_2[U_i]$ for $i \in [k] \setminus \{\sigma\}$ identical to the complement graph $\overline{H}$ (using the same deterministic processes to find H and to choose how to map the vertices).
Bob inserts both edge sets E_σ and E_2, producing final graph $G = (V, E)$. No matter what, the induced subgraph $G[C]$ will be a set of γ isolated vertices guaranteeing that $\alpha(G) \geq \gamma$.

Output Suppose that $X_\sigma = 1$. That means that the induced subgraph $G[U_\sigma]$ consists only of isolated vertices, so $\beta(G) \geq \gamma c\lambda$.
Now suppose $X_\sigma = 0$. That means that every induced subgraph $G[U_i]$ is isomorphic to either H, $\overline{H}$, or the complete graph. So we have $\alpha(G[U_i]) \leq L \log(\gamma c\lambda)$ for every $i \in [k]$. Further, each vertex in each U_i other than U_σ is connected to all vertices from all other U_j's ($j \neq i, \sigma$). So any independent set can contain vertices from only C, U_σ and one other U_i, meaning that $\alpha(G) \leq 2L \log(\gamma c\lambda) + \gamma$
This means that, assuming our algorithm succeeds, in the case $X_\sigma = 1$, the return value is at least $\gamma c\lambda / c_1 = \gamma\lambda c_2$; while in the case $X_\sigma = 0$, the return value is at most $c_2(\gamma + 2L \log(\gamma c\lambda))$.
We can distinguish these cases as long as $\log(\gamma c\lambda) < (\lambda - 1)\gamma/2L$. Since $\lambda > 48L > 2$, we can rewrite this condition as $\log 2\gamma + \log(6Lc^2) + \log(\lambda/12L) < 3 \cdot (\gamma\lambda/12L)$. Recall that $\gamma \geq 1$. Again, since $\lambda/24L > 2$, we know that $\log 2\gamma \leq 2\gamma\lambda/24L$ and $\log(\lambda/12L) < \gamma\lambda/12L$. It is also clear that $\log(6Lc) \leq \gamma\lambda/6L$.

Therefore, we can distinguish between the cases and solve the instance of $\textsc{Index}_k$ with probability at least $\frac{2}{3}$ having sent a message of size at most $B(n)$. By Theorem 2.2.10 it must be that $B(n) = \Omega(k) = \Omega(n/\gamma c \log c)$. □

4.5 Geometric Intersection Graphs

In the previous section, we considered algorithms for arbitrary graphs which gave us a good estimate of $\alpha(G)$ for the broad class of polynomially-bounded dependence graphs.

In this section, we take a more narrow focus. We look at specific classes of geometric intersection graphs, and examine the hardness of approximating maximum independent set and $\alpha(G)$ in explicit and implicit vertex streams restricted to each class.

Recall that, in a geometric intersection graph, each vertex is associated with a geometric object. All these objects are subsets of some shared space. Then any pair of vertices have an edge between them if and only if their respective geometric objects intersect.

We will consider (closed) interval and square intersection graphs (in the real line and the real plane respectively). For the implicit stream representations, we assume that intervals and squares are represented by their centers and their lengths. We assume that the center is a value in $[M]^d$ ($d = 1$ for intervals, and $d = 2$ for squares), and the length is in $[M]$, for some $M \in [n^{\Theta(1)}]$. Then the adjacency function is simply the overlap function which returns 1 when the two shapes overlap and 0 otherwise.

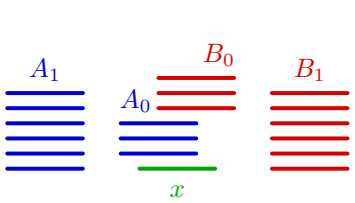
For explicit streams, we simply promise that at least one valid assignment of geometric objects to vertices exists that is consistent with the observed edges. However, we never materialise this representation.

4.5.1 Unit Interval Graphs

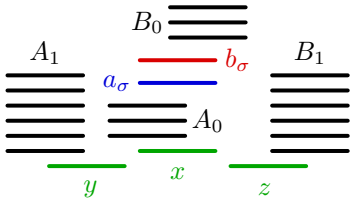
As discussed in Section 4.1.1, given a stream of unit intervals we can compute a $\frac{3}{2}$-approximation to maximum independent set in $\tilde{O}(\alpha(G))$ space, and any better approximation requires $\Omega(n)$ space.

A natural question is how this compares with the space complexity for an interval intersection graph given as an explicit vertex stream:

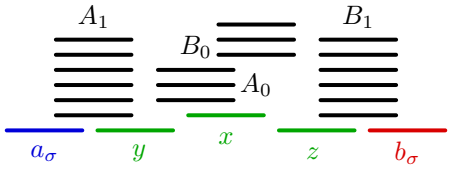
Theorem 4.5.1. *Any $\frac{1}{3}$-error randomised explicit vertex streaming algorithm that returns a $(\frac{5}{3} - \epsilon)$-approximation of $\alpha(G)$ for a unit interval stream must use $\Omega(n)$ bits of space.*



(a) First party makes 3 cliques.



(b) After second party adds y and z, if $X_\sigma = 0$ then $\alpha(G) = 3$.



(c) Conversely, if $X_\sigma = 1$ then $\alpha(G) = 5$.

Figure 4.7: Interval representations for the construction in proof of Theorem 4.5.1. Horizontal positioning represents the location of the intervals in $\mathbb{R}$, vertical positioning is for clarity only.

Proof. We will show this bound by a reduction from the 2-party INDEX communication problem. Consider an instance of INDEX with bit vector $X \in \{0,1\}^n$ and index to be queried $\sigma \in [n]$. We will construct a $2n+3$ vertex graph as an explicit vertex stream.

Label the vertices x, y, z and a_i, b_i for $i \in [n]$. Split the a_i's into two sets based on the bit vector X: $A_1 = \{a_i\}_{X_i=1}$ and $A_0 = \{a_i\}_{X_i=0}$. Similarly let $B_1 = \{b_i\}_{X_i=1}$ and $B_0 = \{b_i\}_{X_i=0}$. Now the first party creates the following subgraph in the stream: a clique consisting of all the vertices in A_1, a second clique made from B_1, and a third clique containing $A_0 \cup B_0 \cup \{x\}$.

So far this represents a valid interval graph, which can be interpreted as three adjacent "stacks" of intervals. Now, the second player adds y with edges to every a_i except a_σ and then adds z with edges to every b_i except b_σ. This can still be viewed as a valid interval graph, but we now require some intervals from each stack to be "shifted" to overlap with the two new intervals.

In the case of $X_\sigma = 0$, the resulting graph has $\alpha(G) = 3$. Otherwise, $\alpha(G) = 5$. Hence, any algorithm giving a better than $\frac{5}{3}$-approximation factor could distinguish them and solve INDEX, and so by Theorem 2.2.10, the complexity is at least $\Omega(n)$ bits. □

This shows that approximating $\alpha(G)$ for interval graphs is strictly more difficult in explicit vertex streams than implicit ones.

4.5.2 Unit Square Graphs

Our first result for two dimensions is a 3-approximation algorithm for maximum independent set on a unit square stream. This is a generalization of the algorithm of Cabello and Pérez-Lantero [CPL17] for unit interval streams: we perform a decomposition of the plane into 2-by-3 strips, similar to their decomposition of the line into length 3 segments.

Theorem 4.5.2. *There is a deterministic 3-approximation implicit vertex streaming algorithm for maximum independent set on a stream of unit squares using $\tilde{O}(\alpha(G))$ space.*

Proof. Let w denote the side length of the squares.
First we look at the problem restricted to squares in the half open strip $[0, 3w) \times [0, 2w)$, referring to the first axis as "left-right", and the second as "up-down". At most 2 non-overlapping closed unit squares can fit fully within the open region, and for them not to overlap one must be left of the other. Hence, by storing the leftmost and rightmost squares seen within the strip, we can determine exactly a maximum independent set in the region.
Now, consider the whole of $[M]^2$. We partition this up into $3w$-by-$2w$ half-open strips and consider only the squares which fall exactly within one of the strips. By solving maximum independent set within each strip as before and taking the union, we can solve exactly maximum independent set on this substream. This requires us to store at most twice as many squares as the solution.
Finally, we consider 6 different copies of the partitioning shifted by 0, w, or $2w$ horizontally and 0 or w vertically. Any square from the stream must be fully contained in a strip in at least 2 of the 6 partitionings. In particular, this holds for every square in a fixed maximum independent set of the full stream. Using $G_{x,y}$ for the graph of the substream of squares that fit exactly in the partitioning shifted by (x, y), we know the following: $\sum_{x=0,w,2w} \left(\sum_{y=0,w} (\alpha(G_{x,y})) \right) \geq 2\alpha(G)$. That is, the sum of the sizes of the substream maximum independent sets for the 6 partitionings is at least 2 times the size of the true maximum independent set. Therefore $\max_{x=0,w,2w} \left(\max_{y=0,w} (\alpha(G_{x,y})) \right) \geq \frac{1}{3}\alpha(G)$
Therefore, we simply take the largest of the 6 substream maximum independent sets and this must give a 3-approximation. In total, we must store at most 12 squares per maximum independent set square. □

As with Cabello, Sergio and Pérez-Lantero [CPL17] for unit intervals, this immediately leads to a sublinear space algorithm for estimating $\alpha(G)$ with only a $(1+\epsilon)$ factor loss in approximation factor, through a combination of counting distinct elements and clever sampling.

Corollary 4.5.3. *There is a $\frac{1}{3}$-error randomised implicit vertex streaming algorithm that can $(3+\epsilon)$-approximate $\alpha(G)$ in a stream of unit squares using $O(\epsilon^{-2}\log\epsilon^{-1}+\log n)$ bits of space.*

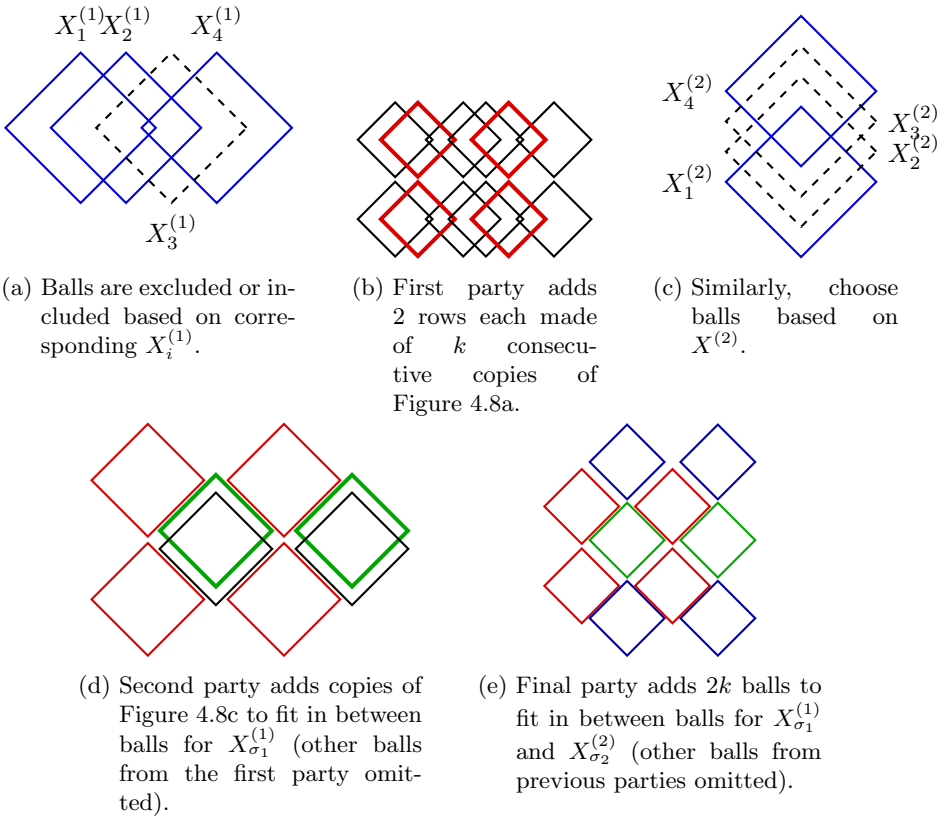
Proof sketch. Observe that if we can get a $(1+\epsilon)$-approximation to each $\alpha(G_{x,y})$ from the proof of Theorem 4.5.2, then we are done.
Each strip can have 0, 1, or 2 disjoint squares in it. To approximate $\alpha(G_{x,y})$ we estimate γ, the number of strips of a given partitioning which are non-empty, and δ, the average number of disjoint squares in the non-empty strips. Then $\alpha(G_{x,y}) \approx \gamma\delta$.
Observe that approximating γ is a distinct elements problem, which we can $(1+\epsilon)$-approximate with constant probability in $O(\epsilon^{-2}+\log n)$ space (see the work of Kane, Nelson, and Woodruff [KNW10a]).
Then δ can be $(1+\epsilon)$-approximated by taking the average over $O(\epsilon^{-2})$ sampled non-empty strips. We obtain these samples using the same permutation trick as [CPL17, Lemma 16], and the analysis of the estimator is also identical. □

One might speculate whether this decomposition approach could afford a better approximation factor based on some different partitioning of the plane. We give evidence for the negative, since any larger strip size results in the fixed-size sub-problems not being solvable exactly, as the following result shows.

Theorem 4.5.4. *Given a stream of w-by-w squares contained in a $(2+\delta)w$-by-$(2+\delta)w$ region, achieving a $(\frac{3}{2}-\epsilon)$-approximation to $\alpha(G)$, with $\frac{1}{3}$-error for any $\epsilon,\delta>0$ requires $\Omega(n)$ space.*

Proof. We show this by a reduction from 2-party Index. Fix an instance with bit vector $X\in\{0,1\}^n$ and query index $\sigma\in[n]$. For the lower bound, we use squares of width $w=\frac{4n}{\delta}$, which meets the requirements for an implicit vertex stream as long as δ is constant.
Party one constructs the following collection of squares arranged along a diagonal line: for each $i\in[n]$ with $X_i=1$ include the square centered on $(\frac{2n}{\delta}+2i, \frac{2n}{\delta}+2n+2-2i)$.Now, observe that the parties could use a $(\frac{3}{2}-\epsilon)$-approximation streaming algorithm to allow party two to determine X_σ. Simply have party one run the algorithm on its collection, then pass the state to party two and append squares centered on $(\frac{6n}{\delta}+2\sigma+1, \frac{2n}{\delta}+2n+2-2\sigma)$ and $(\frac{2n}{\delta}+2\sigma, \frac{6n}{\delta}+2n+3-2\sigma)$. If $X_\sigma=1$, there exists a square in the original collection sandwiched between the two new squares giving $\alpha(G)=3$. Otherwise, $\alpha(G)=2$.
Then by Theorem 2.2.10, the complexity is at least $\Omega(n)$ bits. □

Our next result for two dimensions is a stronger lower bound for approximating $\alpha(G)$ of a stream of unit squares in an unrestricted region, based on a reduction from the



(a) Balls are excluded or included based on corresponding $X_i^{(1)}$.

(b) First party adds 2 rows each made of k consecutive copies of Figure 4.8a.

(c) Similarly, choose balls based on $X^{(2)}$.

(d) Second party adds copies of Figure 4.8c to fit in between balls for $X_{\sigma_1}^{(1)}$ (other balls from the first party omitted).

(e) Final party adds $2k$ balls to fit in between balls for $X_{\sigma_1}^{(1)}$ and $X_{\sigma_2}^{(2)}$ (other balls from previous parties omitted).

Figure 4.8: Example for Theorem 4.5.5 with $X^{(1)} = (1, 1, 0, 1)$, $X^{(2)} = (1, 0, 0, 1)$, $\sigma_1 = 2$, $\sigma_2 = 4$, $k = 2$.

chained index communication problem discussed in Section 4.2 and used in Section 4.3.

Theorem 4.5.5. *Any $\frac{1}{3}$-error randomised implicit vertex streaming algorithm which can $(\frac{5}{2} - \epsilon)$-approximate $\alpha(G)$ on a unit square stream requires $\Omega(n)$ bits of space for any $\epsilon > 0$.*

Proof. This proof works by reducing from the 3-party chained index problem. As with the general result in Section 4.3.2, more than 2 parties are necessary in order to give a bound for approximation factors greater than 2.

Suppose we have an instance of $\textsc{Chain}_3$ with n-bit vectors. We will describe a way for each party to construct a collection of unit l_1 balls based on the part of the input they hold, such that their union has small or large $\alpha(G)$ depending on the solution to the communication problem. Thus, a streaming algorithm for approximating $\alpha(G)$ can be used to solve the communication problem, giving a space bound.

For the construction we use domain size $M = 10n^3$ and ball radius $r = 2n^2$, which are small enough to allow succinct $\operatorname{polylog} n$ sized descriptions of the balls, but

large enough to create the gadgets we require.

Fix integer $k \in [n]$. The first party has the bit vector $X^{(1)}$. For each entry with $X_i^{(1)} = 1$, add balls centered at $(i(4n+3) + (j+1)(4n^2+3n), 4n^2)$ and $(i(4n+3) + (j+1)(4n^2+3n), 8n^2)$ for each $j \in [k]$. Essentially, this makes two horizontal lines of balls stacked on top of each other. There are potentially nk ball locations along the line with centers $4n+3$ apart. The first n locations are associated with the n entries of $X^{(1)}$; we place a ball if $X^{(1)} = 1$, and omit it otherwise; then this is repeated k times in succession. The two lines produce a collection of balls G_1 of size at most $2nk$, with $\alpha(G_1) = 2k$. Importantly, the collection of balls associated with any index $X_i^{(1)} = 1$ forms a maximum independent set.

Party two will obliviously add their own set of balls, such that $\alpha(G)$ will increase exactly when the answer bit $X_{\sigma_1}^{(1)} = 1$. The second party has its bit vector $X^{(2)}$ and the index σ_1 of the answer bit in the first party's bit vector. For each entry with $X_i^{(2)} = 1$ add a ball centered at $(\sigma_1(4n+3) + (j+\frac{3}{2})(4n^2+3n), 6n^2 - n + 2i)$ for each $j \in [k]$. Essentially, this produces k columns of n balls. The columns are spaced with centers n^2+3n apart, lined up to fit in the gaps between the balls corresponding to bit $X_{\sigma_1}^{(1)}$ of the first party (if those balls are present). Each column is constructed as follows: place n balls spaced with centers 1 apart, each associated with one of the entries of $X^{(2)}$, but exclude each ball whose $X_i^{(2)} = 0$. The result is a collection of balls G_2 of size nk with $\alpha(G_2) = k$. Again, the collection of balls associated with any index $X_i^{(1)} = 1$ forms a maximum independent set.

Now, party three will add the final collection of balls G_3. Party three knows σ_1 (this can be appended to any message from player two) and σ_2, the index of the answer bit in party two's bit vector. This party adds balls centered at $(\sigma_1(4n+3) + (j+\frac{3}{2})(4n^2+3n), 10n^2 - n + 2\sigma_2 + 1)$ and $(\sigma_1(4n+3) + (j+\frac{3}{2})(4n^2+3n), 2n^2 - n + 2\sigma_2 - 1)$ for each $j \in [k]$. These balls sit at the top and bottom of each column from party 2 sandwiching the ball corresponding to $X_{\sigma_2}^{(2)}$ (if present). This final collection has an independent set of size $2k$.

At this point we wish to determine $\alpha(G)$ for the union $G = G_1 \cup G_2 \cup G_3$. In the case that $X_{\sigma_1}^{(1)} = X_{\sigma_2}^{(2)} = 1$, we can take the maximum independent set of G_1 associated with $X_{\sigma_1}^{(1)}$ and the maximum independent set of G_2 associated with $X_{\sigma_2}^{(2)}$ along with all the balls in G_3 to form a maximum independent set of G of size $5k$.

However, if $X_{\sigma_1}^{(1)} = X_{\sigma_2}^{(2)} = 0$, then choosing any ball from G_1 (other than the left-most $\sigma_1 - 1$ balls in each row) excludes every ball in a column of G_2 and a ball from G_3. Similar exclusions occur between the other pairs of collections. The result is that the best we can do is to choose a maximum independent set from G_1 corresponding to an index smaller than σ_1 along with 1 ball from each of G_2 and G_3 in the rightmost column, giving a total of $\alpha(G) = 2k + 2$.

The construction is represented in Figure 4.8.

By Theorem 4.2.2. this shows that a streaming algorithm achieving an approximation factor better than $\frac{5k}{2k+2}$ must use $\Omega(n)$ space. This holds for any constant k (just take n large enough to allow that k), giving the result. □

4.5.3 Arbitrary Square Graphs

If we are allowed a combination of large and small balls, we can slightly improve the lower bound up to the maximum possible for a 3-party construction.

Theorem 4.5.6. *Any $\frac{1}{3}$-error randomised implicit vertex streaming algorithm which can $(3-\epsilon)$-approximate $\alpha(G)$ on a stream of squares or arbitrary side lengths requires $\Omega(n)$ bits of space for any $\epsilon > 0$.*

Proof. We adapt the construction from Theorem 4.5.5 as follows: the first party inserts k rows stacked on top of each other, rather than 2, the second party inserts copies of its columns between every consecutive pair of rows from the first party, and the third party places smaller balls in between the columns of party 2 such that they are independent of the squares corresponding to the answer bit but overlap the squares either side.
This results in a gap of $k^2 + k + 1$ to $3k^2$ for the two cases, giving the result. □

5 Maximum Matchings with Bounded Deletions

5.1 Introduction

Maximum matching is a fundamental graph problem with countless applications. In the offline world, there are efficient polynomial-time algorithms to compute exact maximum matchings (and even maximum-weight matchings in weighted graphs). However, in the streaming setting, we have a dichotomy: in an insert-only edge stream a very simple greedy approach gives a good 2-approximation, using $\tilde{O}(n)$ bits of space for an n-vertex graph; on the other hand, to survive the arbitrary edge deletions of a turnstile edge stream, any algorithm must use at least $\tilde{\Omega}(n^2)$ space to achieve the same approximation quality (details in Section 5.1.1).

This quadratic gap poses a natural question: what can we say if *some* deletions are allowed - but not *arbitrary* deletions? Can we parameterise the problem complexity it terms of the amount or type of deletions allowed?

In this chapter, we show that the answer is yes.

We consider variants of the edge streaming and two-party one-way communication models where at most k edges are allowed to be deleted from the graph. We then demonstrate new algorithms for both models with linear space dependence on the number of deletions k.

This is a massive improvement for settings where the average number of deletions per vertex is much smaller than the average degree. For example, on a social network the average user may have 100's of "friends", but may only rarely "unfriend" people. Then our approach can allow maximum matching to be approximated from a summary containing a small amount of data per user, rather than the full graph.

We also develop new communication lower bounds to show that the approximate maximum matching problem requires $\Omega(n^2/c^3)$ bits of communication in the two-party model with unbounded deletions. This generalises an existing lower bound for turnstile edge streams to simple turnstile edge streams. Our hope is that these techniques can be extended to give tight lower bounds for the bounded deletion case.

5.1.1 Background

A matching in a graph is simply any subset of the edges such that no pair of edges in the subset share an end-point. We wish to find matchings of (approximately) the largest possible size.

Definition 5.1.1 (Matchings)**.** *Consider a graph $G = (V, E)$:*

- *A **matching** in G (or E) is a subset of edges $M \subset E$ such that every pair of distinct edges $m_1, m_2 \in M$ are disjoint: $m_1 \cap m_2 = \emptyset$.*
- *A **maximal matching** in G is any matching M such that for every other edge $e \in E \setminus M$ we have that $M \cup \{e\}$ is not a matching.*
- *Let $\mathrm{MM}(G)$ be the maximum size $|M|$ over matchings M in G.*
- *A **maximum matching** in G is any matching of size $\mathrm{MM}(G)$.*
- *A **c-approximate maximum matching** in G is any matching of size at least $\frac{1}{c}\mathrm{MM}(G)$.*

Like with independent sets, a simple greedy algorithm can be used to find a maximal matching. In GreedyMatch we iterate over the edges in any order and greedily add each of them to an initially empty matching as long as doing so would not violate the matching constraint.

Fact 5.1.2. GreedyMatch *always produces a maximal matching since every excluded edge violates the matching constraint.*

Unlike with independent sets, finding a maximal matching is a good way to approximate maximum matching. Consider any maximal matching M and any maximum matching M' in the same graph. Each edge $m \in M'$ must share a vertex with an edge of M otherwise we could add m to M. On the other hand, each edge of M can only share a vertex with two distinct edges of M' otherwise M' would not be a matching. Therefore:

Fact 5.1.3. *Any maximal matching M in any graph G is also a 2-approximate maximum matching in G.*

Since GreedyMatch does not care about the iteration order, it can be directly implemented on insert-only edge streams. We simply need to store the gathered matching M requiring $O(MM(G) \cdot \log n) \leq O(n \log n)$ bits of space.

Proposition 5.1.4. *We can 2-approximate maximum matching for any n-vertex graph G, described by an insert-only edge stream, using $O(MM(G) \cdot \log n)$ bits of space.*

Surprisingly, this is best approximation factor we know how to achieve, even if we allow space as large as $n^{2-\delta}$ for any $\delta > 0$. Kapralov [Kap13] showed that any randomised $\frac{1}{3}$-error insert-only edge streaming algorithm that finds a c-approximate maximum matching for $c > \frac{e}{e-1}$ must use $n^{1+\Omega(1/\log\log n)}$ bits of space (even for bipartite graphs given as an explicit vertex stream). However, the only known streaming strategies for beating the 2-approximation barrier require multiple passes over the stream or require the stream edges to be provided in a uniformly random order (like in the work of Konrad, Magniez, and Mathieu [KMM12]).

When arbitrary deletions are allowed, as in the turnstile edge stream model, the problem becomes much harder. Assadi, Khanna, Li, and Yaroslavtsev [Ass+16] showed that, for $c < \sqrt{n}$ the space complexity of finding a c-approximate maximum matching with constant probability over a turnstile edge stream is $\Theta(n^2/c^3)$ bits, up to logarithmic factors.

One caveat of their result is that it uses the turnstile stream reduction of Li, Nguyen, and Woodruff [LNW14] that we discussed in Section 2.1.4. As a consequence, the lower bound does not necessarily apply to simple turnstile edge streams.

Weighted Matchings

Matchings also have a natural extension to weighted graphs, where each edge of the graph is equipped with a positive real weight.

Definition 5.1.5 (Weighted Matchings)**.** *Consider a weighted graph $G = (V, E)$ where a weighted edge $e \in E$ between vertices $u, v \in V$ with positive weight $w > 0$ is denoted by $e = (\{u, v\}, w)$.*

Using $w(e)$ to indicate the weight of an edge e we have:

- *A **matching** in G (or E) has weight $w(M) = \sum_{m \in M} w(m)$.*
- *Let $\mathrm{MWM}(G)$ be the maximum weight $w(M)$ over matchings M in G.*
- *A **maximum-weight matching** in G is any matching of weight $\mathrm{MWM}(G)$.*
- *A **c-approximate maximum-weight matching** in G is any matching of weight at least $\frac{1}{c}\mathrm{MWM}(G)$.*

Notice that a maximal matching can have arbitrarily small weight even if there are other maximal matchings with arbitrarily large weight. However, we can still approximate maximum weight matching well using a greedy strategy: we just have to enforce that it iterates in order from highest weight edge to lowest weight edge. Call this strategy GreedyWeightMatch.

Fact 5.1.6. GreedyWeightMatch *always produces a 2-approximate maximum-weight matching in G.*

This follows by similar analysis to Fact 5.1.3. Each edge of a maximum weight matching M' must share a vertex with a higher weight edge selected by the algorithm, but each edge selected by the algorithm can only block two distinct edges this way.

We define a variant of edge streams to support weighted graphs.

Definition 5.1.7 (Weighted Edge Stream)**.** *A **weighted edge stream** describing a weighted graph $G = (V, E)$ is an edge stream where each edge has an associated positive weight. The multiset union of all inserted edges minus the multiset of all deleted edges then gives us the edge set of G.*

Like with unweighted edge streams, we do not allow non-existent edges to be deleted: an edge with a particular weight can only be deleted if an edge with that exact weight is currently present.

Unfortunately, since we now need a very specific ordering of the edges, this no longer has a streaming implementation. However, Paz and Schwartzman [PS17] (later simplified and improved by Ghaffari and Wajc [GW19]) showed that by tracking weights on each vertex, and using them to decide which edges to keep, you can $(2 + \epsilon)$-approximate maximum weight matching over insert-only streams using $\tilde{O}(n/\epsilon)$ bits of space.

Bounded Deletions

We define a new variant of the turnstile edge streaming model where the total number of edge deletions is bounded by a parameter k.

Definition 5.1.8 (Bounded Deletion Edge Stream)**.** *A **k-deletion edge stream** describing a graph $G = (V, E)$ is a turnstile edge stream where at most k of the stream updates are edge deletions.*

We use $G_+ = (V, E_+)$ to refer to the graph of all inserted edges and E_- to refer to the multiset of all deleted edges. The final graph $G = (V, E)$ is then given by $E = E_+ \setminus E_-$.

For weighted graphs, we need a more specialised way of measuring the amount of deletions. The key quantity is how much larger the sum of the weights of deleted edges is than the weight of a maximum weight matching.

Definition 5.1.9 (Bounded Deletion Weighted Edge Stream)**.** *An* α***-deletion weighted edge stream*** *describing a weight graph* $G = (V, E)$ *is a turnstile weighted edge stream where the total weight of the deletions is at most* $\alpha \cdot \mathrm{MWM}(G)$*.*

We again use $G_+ = (V, E_+)$ *to refer to the weighted graph of all inserted edges and* E_- *to refer to the multiset of all deleted weighted edges, so* $E = E_+ \setminus E_-$*.*

We also consider a bounded-deletion version of the corresponding 2-party one-way communication model. Working in this model grants us additional algorithmic power compared with streams, but any lower bounds we can show on it immediately also apply to the k-deletion edge stream model.

Definition 5.1.10 (Two-Party Model)**.** *In the* **k*-deletion two-party one-way communication model*** *we have two players: Alice and Bob.*

- *Alice has a simple graph* $G_+ = (V, E_+)$
- *Bob has a set of edge deletions* $E_- \subset E_+$ *with* $|E_-| \leq k$

Alice may send a single message to Bob, then Bob must solve a problem on the difference graph $G = (V, E)$ *where* $E = E_+ \setminus E_-$*.*

We restrict ourselves to simple graphs, since any lower bounds for simple graphs will apply for general graphs also.

5.1.2 Results

Two tables showing complexity bounds in this chapter on the problem of approximating a maximum matching of a graph. Applies for $0 < \epsilon < 2$.

Deterministic Algorithms	Approx.	Complexity	Theorems
For k-deletion edge streams	$(2+\epsilon)$	$\tilde{\mathrm{O}}(n + k/\epsilon)$	5.2.3
For α-deletion weighted edge streams	$(2+\epsilon)$	$\tilde{\mathrm{O}}(\alpha\, n/\epsilon^2)$	5.2.6
For k-deletion edge streams	$c \geq 1$	$\Omega(n/c + k)$	5.2.7

$\frac{1}{3}$-Error Randomised Algorithms	Approx.	Complexity	Theorems
For k-deletion edge streams	$c \geq 8$	$\tilde{\mathrm{O}}(n + k/c)$	5.3.1
For k-deletion two-party model	$c \geq 8$	$\tilde{\mathrm{O}}(n + k/c^2)$	5.3.8
For n^2-deletion two-party model	$c \geq 1$	$\Omega(n^2/c^3)$	5.4.7

5.2 Deterministic Algorithms

We began by demonstrating a deterministic algorithm for bounded-deletion edge streams. The core idea is to take the basic greedy algorithm and make it robust to deletions by adding redundancy.

5.2.1 Constant-Factor Approximation for Unweighted Matchings

Consider a graph described by a k-deletion edge stream. Recall that G_+ is the graph of all inserted edges and G is the final graph. The graphs all have n vertices.

Fact 5.2.1. *If the maximum number of deletions $k \leq \frac{1}{4}\operatorname{MM}(G_+)$, then the standard greedy algorithm (modified to mark any stored edges which are later deleted) gives a 4-approximation using $\mathrm{O}(\operatorname{MM}(G_+) \cdot \log n) \leq \mathrm{O}(n \log n)$ bits of space.*

Proof. This is because we have at least $\frac{1}{2}\operatorname{MM}(G_+)$ edges in the greedy matching, so at most half of these could have been marked as deleted. Simply returning all the unmarked edges gives a matching of size at least $\frac{1}{4}\operatorname{MM}(G_+) \geq \frac{1}{4}\operatorname{MM}(G)$. □

This gives an idea for a deterministic algorithm: if we can somehow store more matchings, we can tolerate more deletions before all the matchings are lost. To find the right way, consider the following fact.

Fact 5.2.2. *Given a stream of edge insertions, if we perform the greedy algorithm on some sub-stream S to obtain a matching M, we can later extend M to obtain a 2-approximation by passing over the stream again and greedily adding edges from the complement of S.*

Proof. This is easily seen as simply simulating the greedy algorithm over an alternative ordering of the stream, where the sub-stream items occur first followed by the remaining items. Any greedy matching on any ordering always gives a 2-approximation. □

We can use this idea to maintain multiple greedy matchings in the following way: we keep multiple "levels" of matching, each selected greedily from edges rejected by the levels below. Each of these matchings will be edge disjoint, and are greedy matchings taken over a sub-stream. Further, we can extend each of them (in the sense of in Fact 5.2.2) without having to take a second pass, because the only edges each level does not see are those stored on the levels below.

Algorithm 3: Deterministic $(2+\epsilon)$-Maximum Matching

```
def Initialise(k, ε):
    τ ← ⌈4k/ε⌉
    L ← 1
    M_1 ← ∅
def Insert(e):
    a ← smallest i where M_i ∪ {e} is a matching
    if a is defined then
        M_a ← M_a ∪ {e}
        if a = L then
            L ← L + 1
            M_L ← ∅
        if Σ_{i=1}^{L-1} |M_i| ≥ τ then
            delete M_L
            L ← L − 1
def Delete(e):
    for i ∈ [L] do
        M_i ← M_i \ {e}
def Output():
    return maximum matching in ⋃_{i∈[L]} M_i
```

Theorem 5.2.3. *For any $\epsilon \leq 2$, there is a deterministic algorithm for k-deletion edge streams which produces a $(2+\epsilon)$-approximation to maximum matching using $\mathrm{O}\left((k/\epsilon + \mathrm{MM}(G_+))\log n\right) \leq \mathrm{O}\left((k/\epsilon + n)\log n\right)$ bits of space.*

Proof. Consider the state of Algorithm 3 at any point during the stream.
Clearly any given level M_i holds at most $\mathrm{MM}(G_+)$ edges, and the bottom $L-1$ levels must contain fewer than $\lceil 4k/\epsilon \rceil$ edges. This gives the stated space bound.
Now we need to see that, at the end of the stream, the stored edges will contain a large enough matching. Observe that we have at least $\lceil 4k/\epsilon \rceil$ edges stored (unless the graph has fewer edges than that, in which case we recover the entire graph). Therefore, after the k deletions, at least one level must have had at most an $\epsilon/4$ fraction of it's stored edges deleted. Call this level a.
Now consider what would happen if we took all the edges stored in levels below a and greedily tried to add them one-by-one into matching M_a. The resulting matching M is a greedy matching of some order of the stream with at most an $\epsilon/4$ of it's edges deleted. Therefore it must be of size at least $(\frac{1}{2} - \frac{\epsilon}{8})\,\mathrm{MM}(G) \geq \mathrm{MM}(G)/(2+\epsilon)$. □

5.2.2 Extension to Weighted Matchings

This strategy can be generalised for the maximum-weight matching problem on α-deletion weighted edge streams. Recall that $\alpha = w(E_-)/\operatorname{MWM}(G)$ where $w(E_-)$ is the total weight of deleted edges. This is equivalent to $k/\operatorname{MM}(G)$ for unweighted bounded-deletion edge streams.

For this extension, we will use an insert-only streaming algorithm as a subroutine. Recall that:

> **Theorem 5.2.4** (Theorem 4.5 in [GW19]). *For any $\epsilon > 0$ there exists an insert-only streaming algorithm for computing a $(2+\epsilon)$-approximation to $\operatorname{MWM}(G)$, using $O\left(n/\epsilon \cdot \log n \cdot \log\left(1/\epsilon\right)\right)$ bits of space.*

Call this algorithm WeightedAlg(ϵ).

> **Remark 5.2.5.** *In particular, this algorithm stores a subset of edges which is guaranteed to contain such an approximate matching. Some received edges may be rejected immediately from the data structure, while others may be initially stored but then rejected at a later update.*

This reject/accept structure allows us to recreate our unweighted scheme using this algorithm as a black-box replacement for the greedy algorithm. In fact, any deterministic algorithm for approximate maximum weight matching that works by maintaining a pool of candidate edges (and possibly some auxiliary information) to select its output from could be used here.

Algorithm 4: Deterministic $(2+O(\epsilon))$-Maximum Weighted Matching

```
def Initialise(α, ε):
    L ← ⌈α/ε⌉
    for i ∈ [L] do
        D_i ← new WeightedAlg(ε)
def Insert(e):
    i ← smallest j where D_j will accept e
    if i is defined then
        insert e into D_i
        if an edge s was dropped by D_i then
            Insert(s)
def Delete(e):
    for i ∈ [L] do
        remove e from D_i
def Output():
    return maximum weight matching of all edges stored in ⋃_{i∈[L]} D_i
```

Theorem 5.2.6. *For any $\epsilon \leq 2$, there is a deterministic algorithm for α-deletion weighted edge streams which produces a $(2+\epsilon)$-approximation to maximum weight matching using $\mathrm{O}\left(\alpha\, n/\epsilon^2 \cdot \log n \cdot log(1/\epsilon)\right)$ bits of space.*

Proof. Consider the state of Algorithm 4 at any point during the stream.
Since our algorithm only stores $\lceil \frac{\alpha}{\epsilon} \rceil$ copies of WeightedAlg(ϵ), we immediately get the space usage from Theorem 5.2.4.
Now we need to see that, at the end of the stream, the stored edges will contain a heavy enough matching. Recall that the total weight of the deletions is at most $\alpha \cdot \mathrm{MWM}(G)$ and we partitioned the stream into $\lceil \frac{\alpha}{\epsilon} \rceil$ levels. So at least one level experienced less than $\epsilon \cdot \mathrm{MWM}(G)$ weight of deletions to its stored edges. Call this level i.
Now consider what would happen if we took all the edges stored in levels below i and fed them one-by-one into D_i. The resulting algorithm state D_i^* would have processed every edge of the graph, but still only had $\epsilon \cdot \mathrm{MWM}(G)$ worth of stored edges deleted. By the approximation guarantee in Theorem 5.2.4, if these deletions had not happened, then the the edges would contain a matching of weight at least $\mathrm{MWM}(G)/(2+\epsilon)$. So we must still have a matching of size at least:

$$\left(\frac{1}{2+\epsilon} - \epsilon\right) \mathrm{MWM}(G) \geq \mathrm{MWM}(G)/(1+\mathrm{O}(\epsilon))$$

A simple substitution of ϵ then lets us a achieve a $(2+\epsilon)$-approximation, with only a constant factor increase in space used. □

5.2.3 Deterministic Lower Bound

To prove matching lower bounds, we consider the k-deletion two-party model.

Theorem 5.2.7. *Any deterministic protocol which c-approximates the maximum matching problem in the k-deletion two-party setting, must communicate at least $\Omega(n/c + k)$ bits. This holds for any $c \geq 1$.*

Proof. Suppose there is a two-party deterministic protocol $\mathcal{A}$ which solves the problem.
Let Alice have a bipartite graph with $n/2$ vertices in each partition, such that each edge is included independently with probability $1/2$. The expected maximum matching size is $\Omega(n)$, since we expect to include half of any particular perfect matching. Alice sends her message to Bob according to $\mathcal{A}$.
Now, Bob will repeatedly run his part of $\mathcal{A}$ pretending to have different deletion sets $E_1, E_2, \cdots$. Initially, he assumes no deletions $E_1 = \emptyset$ and finds matching M_1. He continues simulating deletions $E_i = \bigcup_{j<i} M_j$ to find matching M_i until $|E_i| > k$.

Bob will always be able to learn at least k edges of Alice's graph using this strategy since, as long as $|E_i| \leq k$, we know that simulating $\mathcal{A}$ with deletion set E_i is guaranteed to return at least one edge of the remaining graph (unless the remaining graph is empty, and we have learned everything). Further, the first learned edge set M_1 must be of size at least $\text{MM}(G)/c$, by the approximation guarantee. Over the randomness of the input graph, we learn at least $\Omega(n/c)$ edges in expectation. Each input edge has 1 independent bit of entropy, so Bob learned $\Omega(n/c + k)$ bits of information about the graph in expectation, giving us the communication lower bound. □

This immediately applies also to the bounded-deletion streaming setting.

Corollary 5.2.8. *Any deterministic algorithm for k-deletion edge streams which returns a c-approximation to the maximum matching problem, must communicate at least $\Omega(n/c + k)$ bits.*

Observe that $\alpha = k/n$ for uniformly weighted k-deletion streams which contain a perfect matching, so this shows that both our algorithms are essentially optimal in terms of space cost for deterministic approximation schemes of any quality - though it could be possible to improve dependency on ϵ. The most interesting open question is whether we can beat the 2-approximation barrier in $o(n^2)$ space, which would be big breakthrough even for insert-only edge streams.

5.3 Randomised Algorithms

We saw that the robust greedy approach is essentially optimal for deterministic algorithms, but adding randomisation can often increase algorithmic power. This turns out to be true for maximum matching in that we can achieve $o(k)$ space for worse-than-constant-factor approximations.

5.3.1 Simple Streaming Algorithm

Our first randomised algorithm is a small twist on our deterministic Algorithm 3. By sub-sampling the incoming edges, we can scale down the number of deletions seen at the cost of potentially shrinking the maximum matchings by the same factor. This gives a simple $\tilde{O}(\text{MM}(G) + k/c)$ space streaming algorithm.

Theorem 5.3.1. *For any $c \geq 8$, there is a δ-error randomised algorithm for k-deletion edge streams which produces a c-approximation to maximum matching using $O((\text{MM}(G) + k/c \cdot \log n) \cdot \log \frac{1}{\delta})$ bits of space.*

Proof. Suppose we filter the stream in the following way: each time an edge e is inserted, with probability $8/c$ we process the update according to Algorithm 3 (using $\tau = \lceil 8k/c \rceil$), otherwise we ignore it. If that inserted edge would have been deleted later in the stream, we effectively ignore the deletion too, since the algorithm will simply find that the edge is not present to remove.
Therefore the operation of our algorithm is identical to if we ran on a substream with the inserts and deletions both subsampled.
Now consider the set of edge deletions E_- and a maximum matching M of the full streamed graph. Each edge of both sets is processed independently with probability $8/c$, so the number of deletions in the filtered stream is at most $\lceil 8k/c \rceil$ with probability at least $\frac{1}{2}$ and, independently, the number of edges of M appearing in the substream is at least $\lfloor 8|M|/c \rfloor$ with probability at least $\frac{1}{2}$.
If $|M|/c \leq 1$, then we only need to use an l_0 sampler to find any edge. Otherwise, with probability at least $1/4$, our algorithm receives at most $\lceil 8k/c \rceil$ deletions, so the parameter τ was set correctly for a 3-approximation, and we know the algorithm received a matching of size at least $3|M|/c$, so we find a good enough approximation.
Now, we simply try this strategy $O(\log \delta^{-1})$ times in parallel, to get the correct error guarantee. □

However, we can do better - at least in the two-party setting. In the remainder of this section we work towards demonstrating an $\tilde{O}(\mathrm{MM}(G) + k/c^2)$ space algorithm for the k-deletion two-party model.

We begin by describing a streaming algorithm for graphs with bounded degree d and then show how it can be used to solve the problem for arbitrary graphs in the k-deletion two-party model.

5.3.2 Improved Algorithm for Bounded-Degree Graphs

Suppose that we have a turnstile edge stream (it does not need to be bounded-deletion) which describes a graph $G = (V, E)$ with bounded-degree d. That is, every vertex $v \in V$ has $\deg_G(v) \leq d$. The graph may have higher degree vertices part way through the stream, but the constraint must be satisfied at query time.

A very simple strategy of sampling one edge incident to each vertex can work very well for nearly-regular graphs (where most vertices have close to maximum degree), as shown by the following variation on a well known result (a uniform sample would satisfy $p = 1$):

Lemma 5.3.2. *Let $p > 0$ and let $G = (V, E)$ be a graph with maximum degree $d = \max_{v \in V}(\deg_G(v))$.*
Suppose that, for every vertex $v \in V$, we independently collect a sample of edges E_v such that each edge incident on v is included with probability at least $p/\deg_G(v)$.
Then, with probability at least $\frac{1}{2}$, we have that $\mathrm{MM}\left(\bigcup_{v \in V} E_v\right) \geq \lfloor |E| \cdot p/16d \rfloor$.

Proof. Consider Algorithm 5 applied to G. It starts with an empty matching M' and an empty vertex set V'. It maintains the invariant that V' contains all the end-points of M', so whenever we find an edge with no end-points in V' we can add it to the matching.

As long as V' is smaller than $|E|/2d$, at most $|E|/2$ edges are incident upon V'. This means that at least half the edges of the graph are not incident upon V', so there is at least one vertex $x \in V \setminus V'$ which has at least half of its neighbourhood outside of V'. That is $|N_G(x) \cap V'|/|N_G(x)| \leq \frac{1}{2}$.

In particular, this means that $N_G(v) \setminus V'$ is non-empty for every loop of the algorithm. So in each loop, with probability $p/2$ we find an edge $\{u, v\}$ which can be added to matching M'.

Each loop adds at most 2 vertices to V' and the set is initially empty, so we run for at least $\lceil |E|/4d \rceil$ iterations. Each iteration adds an edge to M' with probability at least $\frac{p}{2}$ regardless of previous successes or failures. This means the cumulative distribution of the number of successes is bounded from below by the cumulative binomial distribution with $\lceil |E|/4d \rceil$ trials and success probability $\frac{p}{2}$.

Such a binomial has median no smaller than $\lfloor |E| \cdot p/8d \rfloor$, so M' is at least this size with probability at least $\frac{1}{2}$.

Now, to prove the lemma, we need to see that this toy algorithm can be simulated on our gathered samples. □

Algorithm 5: Toy Algorithm For Matching of Median Size $\lfloor |E| \cdot p/16d \rfloor$

Input : Graph $G = (V, E)$ and maximum degree $d \leftarrow \max_{v \in V}(\deg(v))$

$V', M' \leftarrow \emptyset$
while $|V'| < |E|/2d$ **do**
  $v \leftarrow \arg\min_{x \in V \setminus V'} (|N_G(x) \cap V'|/|N_G(x)|)$
  $V' \leftarrow V' \cup \{v\}$
  with *probability* $p/2$ **do**
    $u \leftarrow$ any vertex from $N_G(v) \setminus V'$
    $M' \leftarrow M' \cup \{\{u, v\}\}$
    $V' \leftarrow V' \cup \{u\}$
return M'

For example, if at least half the vertices have at least half the maximum degree, then we know $|E| \geq nd/4$. So we could find a matching of size $n/64$ with good probability

by just taking a few samples per vertex.

Now let M be a maximum matching in G and let $V_M = \bigcup_{e \in M} e$ be the set of vertices covered by M. To achieve a more general strategy, we make the following observation: at query time, either the majority of vertices $v \in V_M$ have "low degree" or the majority have "high degree".

In the first case, sampling a random neighbour of each vertex will find us many edges of M in expectation. In the second case, there are enough edges that we can use our good algorithm for nearly-regular graphs from Lemma 5.3.2.

Lemma 5.3.3. *Let $c \geq 1$ and $G = (V, E)$ be a graph with maximum degree $d = \max_{v \in V}(\deg_G(v))$. Suppose that, for each $v \in V$, we collect a sample E_v of $\min(\lceil 64d/c^2 \rceil, \deg_G(v))$ distinct edges selected uniformly at random from the edges adjacent to v. Then $\bigcup_{v \in V} E_v$ contains a matching of size at least $|\operatorname{MM}(G)|/c$ with probability at least $\frac{1}{2}$.*

Proof. Define M to be any maximum matching of G and $V_M = \bigcup_{e \in M} e$ again, then split between the two cases:

1. **If $V_M^{\downarrow} = \{v \in V_M \mid \deg_G(v) \leq 32d/c\}$ has $|V_M^{\downarrow}| \geq |V_M|/2$ then:**
   If $|M|/c \leq 1$ then any sampled edge gives a large enough matching. We will always find at least one edge unless the graph is empty.
   Each vertex $v \in V_M^{\downarrow}$ is adjacent to exactly one edge $e \in M$ and has degree $\deg_G(v) \leq 32d/c$. We have $\min(\lceil 64d/c^2 \rceil, \deg_G(v))$ distinct samples in E_v, so the chance it contains the unique edge e is at least $\frac{64d}{c^2} \cdot \left(\frac{32d}{c}\right)^{-1} = \frac{2}{c}$.
   This means that the cumulative distribution of the number of edges of M found can be bounded below by the binomial cumulative distribution function with $|V_M^{\downarrow}|$ trials and success probability $\frac{2}{c}$.
   The median value of this binomial is at least $\lfloor |V_M^{\downarrow}| \cdot 2/c \rfloor \geq \lfloor |M| \cdot 2/c \rfloor \geq |M| \cdot 2/c - 1 \geq |M|/c$. This means we sample at least $|M|/c$ edges of M with probability at least $\frac{1}{2}$.
2. **If $V_M^{\uparrow} = \{v \in V_M \mid \deg_G(v) > 32d/c\}$ has $|V_M^{\uparrow}| \geq |V_M|/2$ then:**
   The degree lower bound, gives us a lower bound on the number of edges:

$$|E| \geq \sum_{v \in V_M^{\uparrow}} \deg_G(v)/2 > |V_M^{\uparrow}| \cdot 16d/c \geq |M| \cdot 16d/c$$

   From Lemma 5.3.2, we know that a single sample per vertex finds us a matching of size at least $|M|/c$ with probability at least $\frac{1}{2}$.

□

Since this algorithm only requires us to be able to pull a pre-determined number of samples from the neighbourhood of each vertex, we can easily implement it over turnstile edge streams using l_0 samplers.

Theorem 5.3.4. *There is a $\frac{1}{2}$-error randomised algorithm for c-approximating maximum matching in turnstile edge streams which uses $O(nd/c^2 \cdot \log^3 n)$ bits of space.*

Proof. We simply need to store $\lceil 64d/c^2 \rceil$ l_0 samplers per vertex with $\delta = 1/n^3$. As we draw from each sample, we can use linearity to subtract the found edges from future samplers to avoid repetitions. With probability at least $(1 - 1/n)$, all the samplers succeed, and we have enough samples per vertex for Lemma 5.3.3. So we find a large enough matching with probability at least $\frac{1}{2}$.
This requires $O(nd/c^2 \cdot \log^3 n)$ bits of space. □

5.3.3 Extension to Graphs with Large Matching

Now we will see how to relate the bounded-degree problem to a special case of the two-party bounded-deletion problem where graphs are promised to contain a large matching (at least a constant fraction of the vertices). Consider an n-vertex insertion graph $G_+ = (V, E_+)$ and a set of deletion edges $E_- \subset E_+$. We will assume the difference graph $G = (V, E_+ \setminus E_-)$ has a large matching $\mathrm{MM}(G) \geq n/8$.

We will borrow the degree partitioning scheme from Kapralov, Khanna, and Suda [KKS14]. Algorithm 6 describes the process for transforming a graph G into a sequence of $L = \lceil \log n \rceil$ decreasing induced subgraphs $G_1, G_2, \cdots, G_L$.

Algorithm 6: Graph Degree Decomposition

Input: Graph $G = (V, E)$
$L \leftarrow \lceil \log |V| \rceil$
$V_1 \leftarrow V$
$G_1 \leftarrow G$
for $i \in [2, L]$ **do**
  $V_i \leftarrow \{v \in V_{i-1} \mid \deg_{G_{i-1}}(v) \leq n/2^{i-1}\}$
  $G_i \leftarrow G[V_i]$
return $(G_i)_{i=1}^{L}$

This construction works nicely for us, letting us work with vertices in degree bands, one of which is guaranteed to contain a large matching.

Lemma 5.3.5. *Suppose we have an n vertex graph $G = (V, E)$ with large matching $\mathrm{MM}(G) \geq n/8$. Let $(G_i)_{i \in L}$ be the graph sequence produced by running Algorithm 6 on G. For each G_i let (V_i, E_i) be the corresponding vertex and edge sets.*
Then, for any matching M in G, there exists an $a \in [L]$ such that $M \cap E_a$ has more than $|M|/L$ edges incident on $V_a \setminus V_{a-1}$.

Proof. Associate each edge of M with its vertex which is contained in the $V_i \setminus V_{i-1}$ with smaller i (breaking ties arbitrarily). Then we have $|M|$ vertices of interest, so at least $|M|/\log n$ must be contained in some $V_a \setminus V_{a-1}$. Since each of these vertices has its edge of M connected to a higher level, they must be contained in E_a. □

Now, we can guarantee that one of that one of the special subgraphs produced by Algorithm 6 contains a very large matching and either: it has maximum degree $O(\sqrt{k/n})$ - allowing us to use Theorem 5.3.4; or it has a large enough number of post-deletion edges to allow us to use Lemma 5.3.2. In either case, we can achieve a good approximation by maintaining l_0 samplers for each vertex of each special subgraph.

Theorem 5.3.6. *There is a protocol for the k-deletion two-party model which, if the graph contains a matching of size at least $n/8$, returns a matching of size at least $n/(256c\log n)$ with probability $\frac{1}{2}$. This algorithm can be implemented in $O(k/c^2 \cdot \log^3 n)$ bits of space.*

Proof. Alice takes her input graph $G_+ = (V, E_+)$ and runs Algorithm 6 on it to produce the sequence $(G_i)_{i\in[L]}$.

Now for each $i \in [L]$ Alice instantiates the algorithm from Theorem 5.3.4 using $d = 32kL/n$ to guarantee a $c/(\log n)$-approximation for each induced subgraph. This requires $O(kLN/c^2 \cdot \log^3 n)$ bits of space in total.

Alice passes the vertex sets $(V_i)_{i\in[L]}$ and the algorithm states to Bob, who can now apply the appropriate deletions to each subgraph.

Now consider a maximum matching M in $(E_+ \setminus E_-)$. From Lemma 5.3.5 we know that there is an $a \in [L]$ such that at least $|M|/L$ edges of $M \cap E_a$ are incident on $Z_a = V_a \setminus V_{a+1}$. This tell us that $|Z_a| \geq |M|/L$.

We also know that G_a has maximum degree $d_a = n/2^{a-1}$ while the vertices of Z_a have degree at least $n/2^a = d_a/2$ (in G_a). So we can lower bound the number of edges $|E_a| \geq |M| \cdot d_a/2L \geq nd_a/16L$.

Now compare the size of $|E_a|$ with the number of deletions k. There are two cases:

1. **If $|E_a| \geq 2k$ then:**
   The number of post-deletion edges must satisfy $|E_a \setminus E_-| \geq |E_a|/2$ and we know $|E_a|/2 \geq nd_a/32L$. So we have a graph with maximum degree d_a, which contains at least $d_a \cdot (n/32L)$ edges. From Lemma 5.3.2 we know that drawing from a single one of our l_0 samplers per vertex will find us a matching of expected size at least $n/256L \geq n/(256cL)$.
2. **If $|E_a| \leq 2k$ then:**
   Rearranging $2k \geq |E_a| \geq nd_a/16L$ we have $d_a \leq 32kL/n$.
   So we have a bounded degree subgraph which contains a matching of size at least $n/8L$, so the by the guarantee of Theorem 5.3.4, we recover a matching of size $\mathrm{MM}(G)/c$ with probability $\frac{1}{2}$.

□

5.3.4 Extension to Arbitrary Graphs

The work in the previous section allows us to approximate large matchings well, but we would like a more general guarantee. To achieve this, we can make use of Algorithm 7 to reduce the number of vertices without shrinking the matchings, and thus reducing to the situation of having a large matching to find.

Algorithm 7: Randomly Collapsed Graph

Input: Graph $G = (V, E)$ and number of groups γ

$V_H \leftarrow [\gamma]$
$f \leftarrow$ a function mapping V to V_H chosen uniformly at random
$E_H \leftarrow \emptyset$
for $\{u, v\} \in E$ **do**
  $E_H \leftarrow E_H \cup \{f(u), f(v)\}$
$H \leftarrow (V_H, E_H)$
return H

Lemma 5.3.7. *Consider a graph $G = (V, E)$. Suppose we run Algorithm 7 on G for group size γ to produce $H = (V_H, E_H)$. Then:*

- *Any matching M in H implies the existence of a matching of the same size in G. That is, $\mathrm{MM}(G) \geq \mathrm{MM}(H)$.*
- *If $\gamma \leq \mathrm{MM}(G)$, then with probability at least $\frac{1}{2}$, the matching size in H is at least:*

$$\mathrm{MM}(H) \geq \gamma/16$$

Proof. The first property holds because each disjoint pair of edges in H indicate the presence of at least one corresponding pair of disjoint edges in G in order to have inserted them.

Now, choose a matching M in G of size γ. Consider how these edges are mapped into H. Each pair of edges will collide on at least one endpoint with probability at most $2/\gamma$, so the expected number of collision for a given edge is at most 2. By the Markov inequality, each collides with at most 4 others with probability at least $\frac{1}{2}$. By Theorem 2.2.3, at most $\frac{1}{4}$ of the edges collide with at most 4 otherwise, with probability at least $\frac{1}{2}$. And so in this case we can greedily take a quarter of these edges getting $\mathrm{MM}(H) \geq \gamma/16$. □

This suggests a good strategy for dealing with arbitrary matching sizes. We can guess $\lceil \log n \rceil$ different matching sizes of $\gamma = 1, 2, 4, \cdots, n/2, n$ and for each of these, use Algorithm 7 to collapse a copy of the graph to γ vertices. Then we run our large matching recovery algorithm from Theorem 5.3.6 on each collapsed graph. One of these should be the correct choice and return a large matching.

Theorem 5.3.8. *There is a $1/n$-error randomised protocol for the k-deletion two-party one-way communication model, which finds a c-approximate maximum matching using $O(k/c^2 \cdot \log^5 n)$ bits of space for any $c \geq 8$.*

Proof. If $8 \leq c \leq 8192 \log n$, then we revert to the constant factor deterministic algorithm of Theorem 5.2.3.
Run Algorithm 7 on G for $\lceil \log n \rceil$ levels $\gamma = 1, 2, 4, \cdots, n/2, n$. For each of those run the algorithm from Theorem 5.3.6 with approximation factor $c/8192 \log n$.
For one of the guesses γ, we have that $\gamma \leq \text{MM}(G) \leq 2\gamma$, so $\text{MM}(H) \geq \text{MM}(G)/32$ with probability at least $\frac{1}{2}$. Conditioned on this event, the algorithm for that level returns a c-approximation with probability $\frac{1}{2}$.
So we find a c-approximation with probability at least $\frac{1}{4}$. Repeat $O(\log n)$ times in parallel, and we succeed at least once with probability at least $(1 - 1/n)$. Simply return the largest matching from the union of all the outputs. □

5.4 Randomised Lower Bounds

In this section we present work towards lower bounds in the bounded-deletions two-party model. The hope is that these techniques can be extended to give tight bounds for approximate maximum matching with bounded deletions. However, the new lower bounds we demonstrate for the two-party model with unbounded deletions are interesting in their own right.

Before we dive in, we introduce some useful notation for constructing and manipulating matrices.

Definition 5.4.1 (Permutations)**.** *For each integer n:*

- *Let $\mathcal{P}_n$ refer to the set of all permutations mapping $[n] \to [n]$.*
- *Then $\mathcal{P}_n^2$ is the set of functions $\sigma : [n]^2 \to [n]^2$ of the form[a] $\sigma = \sigma_x \times \sigma_y \in \mathcal{P}_n \times \mathcal{P}_n$.*
- *For any binary matrix $M \in \{0,1\}^{n \times n}$ and map $\sigma \in \mathcal{P}_n^2$, we use M_σ to refer to the permuted matrix where entry $(M_\sigma)_{i,j} = M_{\sigma(i,j)}$.*

[a] Recalling that $\sigma_x \times \sigma_y$ refers to the map $(i,j) \mapsto (\sigma_x(i), \sigma_y(j))$.

It will also be useful to have notation for describing parts of matrices, in order to model which bits are known or unknown in our communication problems. To this end - we use a bitwise AND operator to "mask" parts of a matrix. We also make notation to allow us to build masking matrices from conditions on the indices.

Definition 5.4.2 (Masks)**.** *For integers n, $1 \leq k \leq n$, and $0 \leq l \leq k$:*

- *Given any pair of binary matrices $X, Y \in \{0,1\}^{n \times n}$, let $X \wedge Y$ be the entry-wise product of the two matrices: $(X \wedge Y)_{i,j} = X_{i,j} \cdot Y_{i,j}$. This can be seen as the matrix X "masked" by the matrix Y (or vice versa).*
- *Given a logical formula ϕ over variables x and y, let $\textsc{Mask}_n[\phi(x,y)] \in \{0,1\}^{n \times n}$ such that:*

$$(\textsc{Mask}_n[\phi(x,y)])_{i,j} = \begin{cases} 1 & \text{if } \phi(i,j) = \textsc{True} \\ 0 & \text{otherwise} \end{cases}$$

5.4.1 Augmented Bi-Index

Definition 5.4.3. *In an instance of **Augmented Bi-Index** $\textsc{BInd}_\delta^{n,k}$ we have two players - Alice and Bob:*

- *Alice knows a binary matrix $X \in \{0,1\}^{n \times n}$.*
- *Bob knows:*
  - *A pair of permutations on $[n]$ called $\sigma = \sigma_x \times \sigma_y \in \mathcal{P}_n^2$.*
  - *The incomplete binary matrix:*

$$Y = X_\sigma \wedge \textsc{Mask}_n[(x \leq k) \wedge (y \leq k) \wedge ((x,y) \neq (k,k))]$$

The problem is as follows: Alice sends a single message $\mathcal{M}$ to Bob and then Bob must output $X_{\sigma(k,k)}$ with probability at least $(1-\delta)$.

Theorem 5.4.4. *The randomised communication complexity of $\textsc{BInd}_\delta^{n,k}$ is at least $(n-k)^2(1-H(\delta))$.*

The proof relies on two key lemmas.

Lemma 5.4.5. *Suppose that:*

$$\begin{aligned} \mathfrak{A} &= \textsc{Mask}_n[(x > k) \wedge (y > k)] \\ \mathfrak{B} &= \textsc{Mask}_n[(x \leq k) \vee (y \leq k)] \\ \mathfrak{C}_{i,j} &= \textsc{Mask}_n[(i < x \leq k+i) \wedge (j < y \leq k+j) \wedge ((x,y) \neq (k+i,k+j))] \end{aligned}$$

Then for $X \in \{0,1\}^{n \times n}$ chosen uniformly at random we have:

$$H(X \wedge \mathfrak{A} \,|\, X \wedge \mathfrak{B}) \leq \sum_{i,j \in [n-k]} H(X_{k+i,k+j} \,|\, X \wedge \mathfrak{C}_{i,j})$$

Proof. Let:

$$\begin{aligned}
\mathfrak{A}' &= \textsc{Mask}_n\left[(x > k) \wedge (y > k) \wedge ((x, y) \neq (n, n))\right] \\
\mathfrak{B}' &= \textsc{Mask}_n\left[(x < n) \vee (y < n)\right] \\
\mathfrak{C}'_{i,j} &= \textsc{Mask}_n\left[(x \leq k) \vee (y < k + j) \vee ((y = k + j) \wedge (x < k + i))\right]
\end{aligned}$$

Then:

$$\begin{aligned}
H(X \wedge \mathfrak{A} \,|\, X \wedge \mathfrak{B}) &= H(X_{n,n} \,|\, X \wedge \mathfrak{B}') + H(X \wedge \mathfrak{A}' \,|\, X \wedge \mathfrak{B}) && (5.1) \\
&= \sum_{i,j \in [n-k]} H(X_{k+i,k+j} \,|\, X \wedge \mathfrak{C}'_{i,j}) && (5.2) \\
&\leq \sum_{i,j \in [n-k]} H(X_{k+i,k+j} \,|\, X \wedge \mathfrak{C}_{i,j}) && (5.3)
\end{aligned}$$

Line (5.1) holds by the chain rule for conditional entropy, separating out $X_{n,n}$ from the rest of $X \wedge \mathfrak{A}$. By repeating this process separating out with the rest of the column bottom-to-top $X_{n-1,n}, X_{n-2,n}, \cdots, X_{k+1,n}$ before moving on to the next column $X_{n,n-1}, \cdots, X_{k+1,n-1}$ continuing right-to-left, we end up with line (5.2). Finally, line (5.3) holds because shrinking the set of conditions can only increase entropy and $\text{supp}(\mathfrak{C}_{i,j}) \subset \text{supp}(\mathfrak{C}'_{i,j})$.

□

Lemma 5.4.6. *Suppose that we have $X \in \{0,1\}^{n \times n}$ and $\sigma \in \mathcal{P}_n^2$ chosen independently and uniformly at random. Suppose also that:*

$$\mathfrak{C} = \textsc{Mask}_n\left[(x \leq k) \wedge (y \leq k) \wedge ((x, y) \neq (k, k))\right]$$

Then for any deterministic function f:

$$H(X_{\sigma(k,k)} \,|\, f(X), X_\sigma \wedge \mathfrak{C}, \sigma) \geq 1 - \frac{H(f(X))}{(n-k)^2}$$

Proof. We reuse the definitions of $\mathfrak{A}, \mathfrak{B}$, and $\mathfrak{C}_{i,j}$ from Lemma 5.4.5.
Now consider the following collection of permutation pairs:

$$\mathcal{C} = \{\sigma \in \mathcal{P}_n^2 \,|\, \sigma(1,1) = (a(n-k), b(n-k)) \text{ for some integers } a, b\}$$

Observe that every $p \in \mathcal{P}_n^2$ can be uniquely represented as some $c \in \mathcal{C}$ composed with a pair of circular shifts (one of the rows, one of the columns) of size up to $[n-k]$.

Then:

$$
\begin{aligned}
&H(X_{\sigma(k,k)} \mid f(X), X_\sigma \wedge \mathfrak{C}, \sigma) \\
&\qquad = \frac{1}{|\mathcal{P}_n^2|} \sum_{p \in \mathcal{P}_n^2} \left[H(X_{\sigma(k,k)} \mid f(X), X_\sigma \wedge \mathfrak{C}, \sigma = p) \right] && (5.4)\\
&\qquad = \frac{1}{|\mathcal{P}_n^2|} \sum_{p \in \mathcal{P}_n^2} \left[H(X_{p(k,k)} \mid f(X), X_p \wedge \mathfrak{C}) \right] && (5.5)\\
&\qquad = \frac{1}{|\mathcal{P}_n^2|} \sum_{c \in \mathcal{C}} \left[\sum_{i,j \in [n-k]} [H((X_c)_{k+i,k+j} \mid f(X), X_c \wedge \mathfrak{C}_{i,j})] \right] && (5.6)\\
&\qquad \geq \frac{1}{|\mathcal{P}_n^2|} \sum_{c \in \mathcal{C}} [H(X_c \wedge \mathfrak{A} \mid f(X), X_c \wedge \mathfrak{B})] && (5.7)\\
&\qquad \geq \frac{1}{|\mathcal{P}_n^2|} \sum_{c \in \mathcal{C}} [H(X_c, f(X)) - H(f(X)) - H(X_c \wedge \mathfrak{B})] && (5.8)\\
&\qquad \geq \frac{|\mathcal{C}|}{|\mathcal{P}_n^2|} \left[n^2 - H(f(X)) - (n^2 - (n-k)^2) \right] && (5.9)\\
&\qquad = \frac{1}{(n-k)^2} \left[(n-k)^2 - H(f(X)) \right] \\
&\qquad = 1 - \frac{H(f(X))}{(n-k)^2}
\end{aligned}
$$

Line (5.4) holds from the definition of conditional entropy, then line (5.5) follows since X is independent of σ. Line (5.6) simply re-expresses each p as it's unique c, i, j. Now we can use Lemma 5.4.5 applied to each X_c to get line (5.7). Next, line (5.8) holds from the two general properties: $H(A|B,C) = H(A,B,C) - H(B,C)$ and $H(B,C) \leq H(B) + H(C)$. Finally, line (5.9) is true because $H(X_c, f(X)) = H(X, f(X)) = H(X)$. □

With Lemma 5.4.6 we can produce a very short proof of Theorem 5.4.4.

Proof of Theorem 5.4.4. By Yao's principle, we only need to show that there is an input distribution for which any deterministic algorithm must communicate at least $(n-k)^2(1 - H(\delta))$ bits in expectation.

Fix n, k, δ and consider a uniformly sampled instance of $\text{BIND}_{1,\delta}^{n,k}$. Fix a deterministic message function $f : \{0,1\}^{n \times n} \to \mathbb{M}$ which, given Alice's input X, tells us the message Alice with send $f(X)$.

Taking the definition of $\mathfrak{C}$ from Lemma 5.4.6, we can write $Y = X_\sigma \wedge \mathfrak{C}$. Now, by combining Fano's inequality with Lemma 5.4.6 we get:

$$
\begin{aligned}
H(\delta) &\geq H(X_{\sigma(k,k)} \mid f(X), X_\sigma \wedge \mathfrak{C}, \sigma) \\
&\geq 1 - \frac{H(f(X))}{(n-k)^2}
\end{aligned}
$$

which rearrange to give:

$$H(f(X)) \geq (n-k)^2(1 - H(\delta))$$

This gives a corresponding bound on the expected message size. □

5.4.2 Maximum Matching

Theorem 5.4.7. *Any randomised protocol for the n^2-deletion two-party model which finds a c-approximate maximum matching with probability at least $2/3$ must communicate at least $\Omega\left(n^2/c^3\right)$ bits in expectation.*

We will use the following hard distribution:

Definition 5.4.8. *Let $\textsc{Match}_c^n$ be a distribution over instances of the n^2-deletion two-party model, constructed as follows.*
Fix $k = n(1-1/6c)$. Sample independently and uniformly at random a binary matrix $X \in \{0,1\}^{n\times n}$ and a pair of permutations on $[n]$ called $\sigma = \sigma_x \times \sigma_y \in \mathcal{P}_n^2$.
Now consider the two players - Alice and Bob:

- *Alice has the bipartite graph $G = (A \cup B, E_+)$ with $A = \{a_i\}_{i=1}^n$, $B = \{b_j\}_{j=1}^n$, and:*
$$E_+ = \{\{a_i, b_j\} \mid X_{i,j} = 1\}$$
- *Bob has the set of deletions:*
$$E_- = \{\{a_{\sigma_x(i)}, b_{\sigma_y(j)}\} \mid (X_{\sigma(i,j)} = 1) \wedge (i \in [k]) \wedge (j \in [k]) \wedge (i \neq j)\}$$

It will be useful for later to let $D = (X_{\sigma(i,i)})_{i=1}^k$ be the vector representation of the undeleted matching.

Essentially: Alice has a uniformly random bipartite graph with n vertices on each side; then a random perfect matching on $2k$ vertices (k from each side) is chosen, and Bob receives as deletions all the edges in the subgraph induced by those vertices but not contained in the perfect matching.

In this section we will prove Lemma 5.4.9, demonstrating that this is indeed a hard distribution for the problem.

Lemma 5.4.9. *Any deterministic protocol which, with probability at least $2/3$, solves instances sampled from $\textsc{Match}_c^n$ communicates at least $\Omega\left(n^2/c^3\right)$ bits in expectation.*

Which proves the main result.

Proof of Theorem 5.4.7. Follows from Lemma 5.4.9 by Yao's principle. □

But we still have to prove the lemmas.

Proof of Lemma 5.4.9. Fix n, c and consider instances sample from $\textsc{Match}_c^n$. Fix a deterministic message function f which, given Alice's input G, tells us the message Alice will send $f(G)$. Let Z be the set of output edges.
Now suppose that these three inequalities were true:

$$k - \frac{2k}{3c} \geq H(D \mid Z) \tag{5.10}$$

$$\geq H(D \mid f(G), E_{\text{out}}) \tag{5.11}$$

$$\geq k\left(1 - \frac{H(f(G))}{(n-k)^2}\right) \tag{5.12}$$

Rearranging, and recalling that $k = n(1 - \frac{1}{6c})$, we would have:

$$H(f(G)) \geq \Omega\left(\frac{n^2}{c^3}\right)$$

which would also be a lower bound for the expected message size, proving the lemma.
Line (5.11) is simply the data processing inequality. We will prove line (5.10) in Lemma 5.4.10 and line (5.12) in Lemma 5.4.11. □

Lemma 5.4.10. *Suppose we sample an instance of* $\textsc{Match}_c^n$. *Now let* $Z \subset A \times B$ *be a random variable that with probability at least* $\frac{2}{3}$ *is a correct* c*-approximate maximum matching in* $E_+ \setminus E_-$.
Then:

$$H(D \mid Z) \leq k - \frac{2k}{3c}$$

Proof. Let E be an indicator random variable which is 1 with probability exactly $\frac{2}{3}$ and 0 the rest of the time. Choose this such that whenever $E = 1$, Z will be a c-approximate maximum matching.

$$\begin{aligned} H(D \mid Z) &= \mathbb{E}_x\left[H(D \mid Z = x)\right] \\ &\leq \mathbb{E}_x\left[\frac{2}{3}H(D \mid Z = x, E = 1) + \frac{1}{3}H(D)\right] \\ &\leq \frac{2}{3}\left(k - \frac{k}{c}\right) + \frac{k}{3} \\ &= k - \frac{2k}{3c} \end{aligned}$$

□

Lemma 5.4.11. *Suppose we fix a message function f and sample an instance from* $\textsc{Match}_c^n$. *Then:*

$$H(D \mid f(G), E_-) \geq k\left(1 - \frac{H(f(G))}{(n-k)^2}\right)$$

Proof. We show that the conditional entropy is bounded below by k times the entropy in Lemma 5.4.6.
Let:

$$\mathfrak{D}_d = \textsc{Mask}_n\left[(x \leq k) \wedge (y \leq k) \wedge ((x \neq y) \vee (x < d))\right]$$
$$\mathfrak{D}_d^* = \textsc{Mask}_n\left[(x \leq k) \wedge (y \leq k) \wedge ((x, y) \neq (d, d))\right]$$

We can begin by expanding out the entropy using the chain rule for conditional entropy:

$$H(D \mid f(G), E_-) = \sum_{i=1}^{k} H(D_i \mid f(G), E_-, D_{<i})$$

Now consider a particular term of the right hand sum. Since G is exactly determined by the underlying matrix X, there is some function $\tilde{f}$ such that $\tilde{f}(X) = f(G)$ for every sampled instance.

$$\begin{aligned}
H(D_i \mid f(G), E_-, D_{<i}) &= H(D_i \mid \tilde{f}(X), X_\sigma \wedge \mathfrak{D}, D_{<i}) \\
&\geq H(X_{\sigma(i,i)} \mid \tilde{f}(X), X_\sigma \wedge \mathfrak{B}_i) \\
&\geq H(X_{\sigma(i,i)} \mid \tilde{f}(X), X_\sigma \wedge \mathfrak{B}_i, \sigma) \\
&= H(X_{\sigma(k,k)} \mid \tilde{f}(X), X_\sigma \wedge \mathfrak{B}, \sigma) \\
&\geq 1 - \frac{H(\tilde{f}(X))}{(n-k)^2}
\end{aligned}$$

So:

$$H(D \mid f(G), E_-) \geq k\left(1 - \frac{H(\tilde{f}(X))}{(n-k)^2}\right)$$

□

6 Further Questions

We demonstrated various new upper and lower bounds for our three streaming problems, advancing the state of the art. Here we briefly highlight some of the interesting further questions we were not able to address.

Correlation Outliers

- Are there alternative approaches which would allow us to achieve sub-quadratic query times for small outlier thresholds without requiring such large sketches?
- Can our techniques be applied to other measures of distance and similarity?

Independent Sets

- What is the multi-pass complexity of maximal independent set in insert-only edge streams?
- What is the exact communication complexity of the chained-index problem and does it have applications beyond the maximum independent set problem?
- What is the exact complexity of c-approximating $\alpha(G)$ in explicit and implicit vertex streams for general graphs?
- Can we achieve better than (poly $\log n$)-approximation of $\alpha(G)$ for any classes of geometric intersection graph given as *explicit* vertex streams using sublinear space?
- What is the best approximation factor possible for $\alpha(G)$ of implicit unit square streams using sublinear space?
- Is there a constant-factor approximation algorithm for $\alpha(G)$ of implicit arbitrary size square streams using sublinear space?
- Can we extend these results to higher dimensions and other geometric shapes?
- What can we say about the complexity of $\alpha(G)$ approximation for other kinds of special graph classes, such as P_4-free graphs, in each of our graph streaming models?

Maximum Matchings with Bounded Deletions

- Can we break the 2-approximation barrier even for insert-only edge streams without random order streams or multiple passes?
- Is there a streaming version of our bounded-deletion two-party randomised protocol?
- Can our new two-party lower bound techniques be adapted for the bounded-deletion case?
- Does the augmented bi-index problem lead to new lower bounds for other graph problems?[1]
- Can we parameterise the complexity of other graph streaming problems in terms of the amount or type of deletions allowed?

[1] We have since shown in [DK20] that our technique can be adapted to give a tight lower bound for the problem of approximating minimum vertex cover in the two-party model.

Bibliography

[AC06] Nir Ailon and Bernard Chazelle. "Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform". In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. 2006, pp. 557–563.

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. "Sublinear algorithms for $(\Delta+1)$ vertex coloring". In: *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2019, pp. 767–786.

[AG13] Kook Jin Ahn and Sudipto Guha. "Linear programming in the semi-streaming model with application to the maximum matching problem". In: *Information and Computation* (2013), pp. 59–79.

[AGM12] Kook J. Ahn, Sudipto Guha, and Andrew McGregor. "Analyzing graph structure via linear measurements". In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*. 2012, pp. 459–467.

[Ai+16] Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. "New characterizations in turnstile streams with applications". In: *31st Conference on Computational Complexity*. 2016, 20:1–20:22.

[Alo+02] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. "Tracking join and self-join sizes in limited storage". In: *Journal of Computer and System Sciences* (2002), pp. 719–747.

[AMS12] Noga Alon, Ankur Moitra, and Benny Sudakov. "Nearly complete graphs decomposable into large induced matchings and their applications". In: *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*. 2012, pp. 1079–1090.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. "The space complexity of approximating the frequency moments". In: *Journal of Computer and System Sciences* (1999), pp. 137–147.

[AR15] Alexandr Andoni and Ilya Razenshteyn. "Optimal data-dependent hashing for approximate near neighbors". In: *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*. 2015, pp. 793–801.

[Ass+16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. "Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model". In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2016, pp. 1345–1364.

[Ass+19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. "Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs". In: *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2019, pp. 1616–1635.

[BC15] Vladimir Braverman and Stephen R. Chestnut. "Universal sketches for the frequency negative moments and other decreasing streaming sums". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2015, pp. 591–605.

[BC17] Suman K. Bera and Amit Chakrabarti. "Towards tighter space bounds for counting triangles and other substructures in graph streams". In: *34th Symposium on Theoretical Aspects of Computer Science*. 2017.

[BCW19] Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. *Weighted Maximum Independent Set of Geometric Objects in Turnstile Streams*. 2019. arXiv: `1902.10328 [cs.DS]`.

[BDN15] Jean Bourgain, Sjoerd Dirksen, and Jelani Nelson. "Toward a unified theory of sparse dimensionality reduction in euclidean space". In: *Geometric and Functional Analysis* (2015), pp. 1009–1088.

[BG18] Suman Kalyan Bera and Prantar Ghosh. "Coloring in Graph Streams". In: (2018). arXiv: `1807.07640 [cs.DS]`.

[Bhu+06] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. "Simpler algorithm for estimating frequency moments of data streams". In: *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2006, pp. 708–713.

[BKY20] Vladimir Braverman, Robert Krauthgamer, and Lin F. Yang. *Universal Streaming of Subset Norms*. 2020. arXiv: `1812.00241 [cs.DS]`.

[Bła+17] Jarosław Błasiok, Vladimir Braverman, Stephen R. Chestnut, Robert Krauthgamer, and Lin F. Yang. "Streaming symmetric norms via measure concentration". In: *Proceedings of the 49th Annual ACM Symposium on Theory of Computing*. 2017, pp. 716–729.

[Bol88] Béla Bollobás. "The chromatic number of random graphs". In: *Combinatorica* (1988), pp. 49–55.

[BOR15] Vladimir Braverman, Rafail Ostrovsky, and Alan Roytman. "Zero-one laws for sliding windows and universal sketches". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2015.

[Bra+18] Vladimir Braverman, Zaoxing Liu, Tejasvam Singh, NV Vinodchandran, and Lin F. Yang. "New bounds for the CLIQUE-GAP problem using graph decomposition theory". In: *Algorithmica* (2018), pp. 652–667.

[BY+04a] Ziv Bar-Yossef, Thathachar S. Jayram, Robert Krauthgamer, and Ravi Kumar. "The sketching complexity of pattern matching". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2004, pp. 261–272.

[BY+04b] Ziv Bar-Yossef, Thathachar S. Jayram, Ravi Kumar, and D. Sivakumar. "An information statistics approach to data stream and communication complexity". In: *Journal of Computer and System Sciences* (2004), pp. 702–732.

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. "Reductions in streaming algorithms, with an application to counting triangles in graphs". In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2002, pp. 623–632.

[Car79] Yair Caro. *New results on the independence number*. Tech. rep. Tel-Aviv University, 1979.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". In: *Proceedings of 29th International Colloquium on Automata, Languages, and Programming*. 2002, pp. 693–703.

[CCM07] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. "A near-optimal algorithm for computing the entropy of a stream". In: *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2007, pp. 328–335.

[CD18] Graham Cormode and Jacques Dark. "Fast Sketch-based Recovery of Correlation Outliers". In: *Proceedings of the 21st International Conference on Database Theory*. 2018, 13:1–13:18.

[CDK18] Graham Cormode, Jacques Dark, and Christian Konrad. "Approximating the Caro-Wei bound for independent sets in graph streams". In: *Proceedings of 5th International Symposium on Combinatorial Optimization*. 2018, pp. 101–114.

[CDK19] Graham Cormode, Jacques Dark, and Christian Konrad. "Independent Sets in Vertex-Arrival Streams". In: *Proceedings of 46th International Colloquium on Automata, Languages, and Programming*. 2019, 45:1–45:14.

[CH08] Graham Cormode and Marios Hadjieleftheriou. "Finding frequent items in data streams". In: *Proceedings of the VLDB Endowment* (2008), pp. 1530–1541.

[Cha07] Amit Chakrabarti. "Lower bounds for multi-player pointer jumping". In: *22nd Annual IEEE Conference on Computational Complexity*. 2007, pp. 33–45.

[Chi+14] Rajesh Chitnis, Graham Cormode, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. "Parameterized streaming: Maximal matching and vertex cover". In: *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2014, pp. 1234–1251.

[Chi+15] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. *Kernelization via sampling with applications to dynamic graph streams*. 2015. arXiv: `1505.01731 [cs.DS]`.

[Chi+16] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. "Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams". In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2016, pp. 1326–1344.

[CJ17] Graham Cormode and Hossein Jowhari. "A second look at counting triangles in graph streams (Revised)". In: *Theoretical Computer Science* (2017), pp. 22–30.

[CKS03] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. "Near-optimal lower bounds on the multi-party communication complexity of set disjointness". In: *18th Annual IEEE Conference on Computational Complexity*. 2003, pp. 107–117.

[CPL17] Sergio Cabello and Pablo Pérez-Lantero. "Interval selection in the streaming model". In: *Theoretical Computer Science* (2017), pp. 77–96.

[CR12] Amit Chakrabarti and Oded Regev. "An optimal lower bound on the communication complexity of gap-hamming-distance". In: *SIAM Journal on Computing* (2012), pp. 1299–1317.

[CS14] Michael Crouch and Daniel M. Stubbs. "Improved streaming algorithms for weighted matching, via unweighted matching". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2014, pp. 96–104.

[CW09] Kenneth L. Clarkson and David P. Woodruff. "Numerical linear algebra in the streaming model". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. 2009, pp. 205–214.

[CW17] Kenneth L. Clarkson and David P. Woodruff. "Low-rank approximation and regression in input sparsity time". In: *Journal of the ACM* (2017), pp. 1–45.

[CY17] Yu Chen and Ke Yi. "Two-level sampling for join size estimation". In: *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. 2017, pp. 759–774.

[DJS98] Carsten Damm, Stasys Jukna, and Jiří Sgall. "Some bounds on multiparty communication complexity of pointer jumping". In: *Computational Complexity* (1998), pp. 109–127.

[DK20] Jacques Dark and Christian Konrad. "Optimal Lower Bounds for Matching and Vertex Cover in Dynamic Graph Streams". In: *35th Computational Complexity Conference*. 2020, 30:1–30:14.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[EHR16] Yuval Emek, Magnús M. Halldórsson, and Adi Rosén. "Space-constrained interval selection". In: *ACM Transactions on Algorithms* (2016), 51:1–51:32.

[Eps+11] Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. "Improved approximation guarantees for weighted matching in the semi-streaming model". In: *SIAM Journal on Discrete Mathematics* (2011), pp. 1251–1265.

[Far+20] Alireza Farhadi, MohammadTaghi Hajiaghayi, Tung Mah, Anup Rao, and Ryan A. Rossi. "Approximate maximum matching in random streams". In: *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2020, pp. 1773–1785.

[Fei+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. "On graph problems in a semi-streaming model". In: *Theoretical Computer Science* (2005), pp. 207–216.

[Fel+20] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. "The one-way communication complexity of submodular maximization with applications to streaming and robustness". In: *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*. 2020, pp. 1363–1374.

[FIS08] Gereon Frahling, Piotr Indyk, and Christian Sohler. "Sampling in dynamic data streams and applications". In: *International Journal of Computational Geometry & Applications* (2008), pp. 3–28.

[FK14] Stefan Fafianie and Stefan Kratsch. "Streaming kernelization". In: *International Symposium on Mathematical Foundations of Computer Science*. 2014, pp. 275–286.

[FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. "Fast Monte-Carlo algorithms for finding low-rank approximations". In: *Journal of the ACM* (2004), pp. 1025–1041.

[FM85] Philippe Flajolet and G. Nigel Martin. "Probabilistic counting algorithms for data base applications". In: *Journal of Computer and System Sciences* (1985), pp. 182–209.

[Gal14] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. 2014, pp. 296–303.

[Gam+19] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. "Weighted matchings via unweighted augmentations". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 2019, pp. 491–500.

[Gha+16] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. "Frequent directions: Simple and deterministic matrix sketching". In: *SIAM Journal on Computing* (2016), pp. 1762–1792.

[Gil+12] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. "Approximate sparse recovery: optimizing time and measurements". In: *SIAM Journal on Computing* (2012), pp. 436–453.

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. "On the communication and streaming complexity of maximum bipartite matching". In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*. 2012, pp. 468–485.

[GM75] Geoffrey R. Grimmett and Colin J. H. McDiarmid. "On colouring random graphs". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. 1975, pp. 313–324.

[GW19] Mohsen Ghaffari and David Wajc. "Simplified and space-optimal semi-streaming (2+ epsilon)-approximate matching". In: *2nd Symposium on Simplicity in Algorithms*. 2019, 13:1–13:8.

[Hal+10] Bjarni V. Halldórsson, Magnús M. Halldórsson, Elena Losievskaja, and Mario Szegedy. "Streaming algorithms for independent sets". In: *Proceedings of 37th International Colloquium on Automata, Languages, and Programming*. 2010, pp. 641–652.

[Hal+12] Magnús M. Halldórsson, Xiaoming Sun, Mario Szegedy, and Chengu Wang. "Streaming and communication complexity of clique approximation". In: *Proceedings of 39th International Colloquium on Automata, Languages, and Programming*. 2012, pp. 449–460.

[Hås96] Johan Håstad. "Clique is hard to approximate within $n^{1-\epsilon}$". In: *Proceedings of 37th Conference on Foundations of Computer Science*. 1996, pp. 627–636.

[HK15] Magnús M. Halldórsson and Christian Konrad. "Distributed large independent sets in one round on bounded-independence graphs". In: *Proceedings of 29th International Symposium on Distributed Computing*. 2015, pp. 559–572.

[Hoe94] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *The Collected Works of Wassily Hoeffding*. 1994, pp. 409–426.

[HRR98] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. "Computing on data streams." In: *External memory algorithms* (1998), pp. 107–118.

[IM98] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. 1998, pp. 604–613.

[Ind06] Piotr Indyk. "Stable distributions, pseudorandom generators, embeddings, and data stream computation". In: *Journal of the ACM* (2006), pp. 307–323.

[IW03] Piotr Indyk and David P. Woodruff. "Tight lower bounds for the distinct elements problem". In: *44th Annual IEEE Symposium on Foundations of Computer Science*. 2003, pp. 283–288.

[IW05] Piotr Indyk and David Woodruff. "Optimal approximations of the frequency moments of data streams". In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. 2005, pp. 202–208.

[JST11] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. "Tight bounds for lp samplers, finding duplicates in streams, and related problems". In: *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 2011, pp. 49–58.

[JW18] Rajesh Jayaram and David P. Woodruff. "Data Streams with Bounded Deletions". In: *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 2018, pp. 341–354.

[Kan+11] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. "Fast moment estimation in data streams in optimal space". In: *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*. 2011, pp. 745–754.

[Kan+12] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. "Counting arbitrary subgraphs in data streams". In: *Proceedings of 39th International Colloquium on Automata, Languages, and Programming*. 2012, pp. 598–609.

[Kap13] Michael Kapralov. "Better bounds for matchings in the streaming model". In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2013, pp. 1679–1697.

[Kar72] Richard M. Karp. "Reducibility among combinatorial problems". In: *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.

[KKK18] Matti Karppa, Petteri Kaski, and Jukka Kohonen. "A faster subquadratic algorithm for finding outlier correlations". In: *ACM Transactions on Algorithms* (2018), pp. 1–26.

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. "Approximating Matching Size from Random Streams". In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2014, pp. 734–751.

[KLL16] Zohar Karnin, Kevin Lang, and Edo Liberty. "Optimal quantile approximation in streams". In: *57th Annual IEEE Symposium on Foundations of Computer Science*. 2016, pp. 71–78.

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. "Maximum matching in semi-streaming with few passes". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2012, pp. 231–242.

[KNR95] Ilan Kremer, Noam Nisan, and Dana Ron. "On randomized one-round communication complexity". In: *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*. 1995, pp. 596–605.

[Knu69] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 1969.

[KNW10a] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. "An optimal algorithm for the distinct elements problem". In: *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 2010, pp. 41–52.

[KNW10b] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. "On the exact space complexity of sketching and streaming small norms". In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*. 2010, pp. 1161–1178.

[Kon15] Christian Konrad. "Maximum matching in turnstile streams". In: *Proceedings of the 23rd Annual European Symposium on Algorithms*. 2015, pp. 840–852.

[Kon18] Christian Konrad. "A simple augmentation method for matchings with applications to streaming algorithms". In: *43rd International Symposium on Mathematical Foundations of Computer Science*. 2018, 74:1–74:16.

[KS92] Bala Kalyanasundaram and Georg Schintger. "The probabilistic communication complexity of set intersection". In: *SIAM Journal on Discrete Mathematics* (1992), pp. 545–557.

[KT17] Sagar Kale and Sumedh Tirodkar. "Maximum Matching in Two, Three, and a Few More Passes Over Graph Streams". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2017, 15:1–15:21.

[Lei+18] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. "Query optimization through the looking glass, and what we found running the Join Order Benchmark". In: *The VLDB Journal* (2018), pp. 643–668.

[Li08] Ping Li. "Estimators and tail bounds for dimension reduction in l_α ($0 < \alpha \leq 2$) using stable random projections". In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2008, pp. 10–19.

[LNW14] Yi Li, Huy L Nguyen, and David P. Woodruff. "Turnstile streaming algorithms might as well be linear sketches". In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. 2014, pp. 174–183.

[McG05] Andrew McGregor. "Finding graph matchings in data streams". In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*. 2005, pp. 170–181.

[McG14] Andrew McGregor. "Graph stream algorithms: a survey". In: *ACM SIGMOD Record* (2014), pp. 9–20.

[MM13] Xiangrui Meng and Michael W. Mahoney. "Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. 2013, pp. 91–100.

[MP80] J. Ian Munro and Mike S. Paterson. "Selection and sorting with limited storage". In: *Theoretical Computer Science* (1980), pp. 315–323.

[Mut05] Shanmugavelayutham Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc, 2005.

[MW10] Morteza Monemizadeh and David P. Woodruff. "1-pass relative-error L_p-sampling with applications". In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*. 2010, pp. 1143–1160.

[NN13] Jelani Nelson and Huy L. Nguyên. "OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings". In: *54th Annual IEEE Symposium on Foundations of Computer Science*. 2013, pp. 117–126.

[NW10] Jelani Nelson and David P. Woodruff. "Fast manhattan sketches in data streams". In: *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 2010, pp. 99–110.

[Pag13] Rasmus Pagh. "Compressed matrix multiplication". In: *ACM Transactions on Computation Theory* (2013), 9:1–9:17.

[PS17] Ami Paz and Gregory Schwartzman. "A (2+ϵ)-approximation for maximum weight matching in the semi-streaming model". In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2017, pp. 2153–2161.

[RT08] Vladimir Rokhlin and Mark Tygert. "A fast randomized algorithm for overdetermined linear least-squares regression". In: *Proceedings of the National Academy of Sciences* (2008), pp. 13212–13217.

[RV07] Mark Rudelson and Roman Vershynin. "Sampling from large matrices: An approach through geometric functional analysis". In: *Journal of the ACM* (2007), pp. 21–39.

[Sar06] Tamas Sarlos. "Improved approximation algorithms for large matrices via random projections". In: *47th Annual IEEE Symposium on Foundations of Computer Science*. 2006, pp. 143–152.

[SS96] Michael Sipser and Daniel A. Spielman. "Expander codes". In: *IEEE Transactions on Information Theory* (1996), pp. 1710–1722.

[TZ04] Mikkel Thorup and Yin Zhang. "Tabulation based 4-universal hashing with applications to second moment estimation." In: *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2004, pp. 615–624.

[Val12] Gregory Valiant. "Finding correlations in subquadratic time, with applications to learning parities and juntas". In: *53rd Annual IEEE Symposium on Foundations of Computer Science*. 2012, pp. 11–20.

[Ven+15] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P. Chakkappen. "Join size estimation subject to filter conditions". In: *Proceedings of the VLDB Endowment* (2015), pp. 1530–1541.

[VY11] Elad Verbin and Wei Yu. "The streaming complexity of cycle counting, sorting by reversals, and other problems". In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. 2011, pp. 11–25.

[Wan+13] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. "Quantiles over data streams: An experimental study". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, pp. 737–748.

[Wei81] Victor K. Wei. *A lower bound on the stability number of a simple graph*. Tech. rep. Bell Laboratories Technical Memorandum 81-11217-9, Murray Hill, NJ, 1981.

[Woo04] David P. Woodruff. "Optimal space lower bounds for all frequency moments". In: *SODA*. 2004, pp. 167–175.

[Woo14] David P. Woodruff. "Sketching as a tool for numerical linear algebra". In: *Foundations and Trends in Theoretical Computer Science* (2014), pp. 1–157.

[Yao79] Andrew C. Yao. "Some complexity questions related to distributed computing". In: *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. 1979, pp. 209–213.

[Zel12] Mariano Zelke. "Weighted matching in the semi-streaming model". In: *Algorithmica* (2012), pp. 1–20.

[Zuc07] David Zuckerman. "Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number". In: *Theory of Computing* (2007), pp. 103–128.