Separations and Equivalences between Turnstile Streaming and Linear Sketching

John Kallaugher* UT Austin

Eric Price* UT Austin

April 16, 2020

Abstract

A longstanding observation, which was partially proven in [LNW14, AHLW16], is that any turnstile streaming algorithm can be implemented as a linear sketch (the reverse is trivially true). We study the relationship between turnstile streaming and linear sketching algorithms in more detail, giving both new separations and new equivalences between the two models.

It was shown in [LNW14] that, if a turnstile algorithm works for arbitrarily long streams with arbitrarily large coordinates at intermediate stages of the stream, then the turnstile algorithm is equivalent to a linear sketch. We show separations of the opposite form: if either the stream length or the maximum value of the stream are substantially restricted, there exist problems where linear sketching is exponentially harder than turnstile streaming.

A further limitation of the [LNW14] equivalence is that the turnstile sketching algorithm is neither explicit nor uniform, but requires an exponentially long advice string. We show how to remove this limitation for deterministic streaming algorithms: we give an explicit small-space algorithm that takes the streaming algorithm and computes an equivalent module.

*This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

The study of streaming algorithms is concerned with the following question: given a very large dataset that appears over time, what questions can one answer about it without ever storing it in its entirety? Formally, one receives $x \in \mathbb{Z}^n$ (e.g, the indicator vector for the set of edges in a graph) as a series of updates $x_i \leftarrow x_i + \Delta$ (e.g., edge insertions and deletions). One would like to estimate properties of the final vector x while only ever using $o(n)$ space, ideally $\text{poly}(\log n)$. The space used by the algorithm is the primary quantity of interest; other parameters such as update or recovery time are often well-behaved as a matter of course for small-space algorithms. In this paper we focus on 'turnstile' streams, where Δ can be negative, as opposed to insertion-only streams, where it must be positive.

The study of turnstile streaming has been very successful at revealing new algorithmic techniques and insights. It has found wide applicability, with algorithms for a huge variety of problems. Examples include norm estimation in ℓ_2 [AMS96] or other ℓ_p [Ind06, CDIM03]; ℓ_0 sampling [FIS08]; heavy hitters [CCF02, CM05]; coresets for k-median [FS05, IP11]; and graph problems such as finding spanning forests [AGM12], spectral sparsifiers [KLM$^+$14], matchings [AKLY16], and triangle counting [TKMF09, PT12, KP17].

Remarkably, for every single problem described above, the best known algorithm is a *linear sketch*, where the state of the algorithm at time t is given by a linear function of the updates seen to x before time t. And for most of these problems, we know that the linear sketch is optimal.

Linear sketches have a number of other nice properties. Their additivity means that one can, for example, split a data stream across multiple routers and sketch the pieces independently. This has also made such sketches useful in non-streaming applications such as distributed computing [KKM13]. Their output depends only on the final value of x, so they will work regardless of the length of the stream, the order in which the stream arrives, and the intermediate states reached by the stream. Their indifference to stream order means the randomness they use can often be implemented with Nisan's PRG [Nis92, Ind06].

They are also easier to prove lower bounds against, either using the simultaneous message passing (SMP) model (e.g., [Kon15, AKLY16, KKP18]) or additional properties of linearity [PW12].

So it would be nice if every turnstile streaming algorithm could be implemented as a linear sketch. And this *is* true, as shown in [LNW14], but only subject to fairly strong limitations. In this paper, we explore the relationship in more detail. First, we show that some of the [LNW14] limitations are necessary: we present natural problems with large, *exponential* separations between turnstile streaming and linear sketching with the limitations removed. Second, we show how to remove other [LNW14] limitations for deterministic functions.

Separations between turnstile streaming and linear sketching. The result in [LNW14] requires that, in order for a turnstile streaming algorithm to be equivalent to a linear sketch, the streaming algorithm must be able to tolerate *extremely* long streams (longer than 2^{2^n}) that reach correspondingly large intermediate states. In [AHLW16], it was shown that this equivalence can be extended to 'strict' turnstile streams, where the intermediate states never become negative but must still be allowed to become extremely large in the positive direction. However, the result still leaves open the possibility of problems that require $\text{poly}(n)$ space in linear sketching, but in turnstile streaming can be solved in $O(\text{poly}(\log n, \log\log L))$ space for length-L streams, or in $O(\text{poly}(\log n, \log M))$ space for streams whose intermediate state never leave $[-M, M]^n$ (a 'box

constraint').

Such a box constraint is particularly natural in graph streaming: if the stream represents insertions and deletions of edges in a graph, then the intermediate states x should lie in $\{0,1\}^{\binom{n}{2}}$. At the same time, graph streaming is where a theorem on equivalence between streaming and sketching would be most useful: most of the problems for which we have lower bounds on linear sketches but not turnstile streaming involve graphs. The [LNW14] equivalence gives lower bounds for these problems, but only for turnstile algorithms that are indifferent to stream length and tolerate multigraphs at intermediate stages.

The conjunction here, where the box constraint is most relevant in precisely the situations where we have no alternative lower bounds to [LNW14], suggests an opportunity: perhaps we have not found direct turnstile streaming lower bounds for these problems because no such lower bounds exist that respect the natural constraints of graphs. Maybe better algorithms exist, and we just haven't found them because they require substantially different, nonlinear approaches to turnstile sketching.

In this paper, we show that this can in fact be the case, presenting natural assumptions on adversarially ordered turnstile streams for which we can prove exponential separations between turnstile streaming and linear sketching. We give several different settings in which there are problems that can be solved with an $O(\log n)$ space streaming algorithm, but for which any linear sketch requires $\widetilde{\Omega}(n^{1/3})$ space.

We first consider *binary* streams: the data stream can have arbitrary length, but must lie in $\{0,1\}^n$ at all times. We present a problem that can be solved over such streams in $O(\log n)$ space, but requires $\Omega(n^{1/3}/\log^{1/3} n)$ space to solve in linear sketching.

We then consider *short* streams: the data stream can have arbitrary intermediate states, but only $L = O(n)$ updates. We show that for a natural problem—triangle counting on bounded degree graphs with many triangles—an $O(\log n)$ space streaming algorithm is possible, while any linear sketching algorithm takes $\Omega(n^{1/3})$ space. The streaming algorithm depends polynomially on L, and a separation remains for any $L = o(n^{7/6})$.

The only previously known separation between turnstile streaming and linear sketching is due to Jayaram and Woodruff [JW18], which for ℓ_1 estimation with $L = O(n)$ gives a separation of $O(\log \log n)$ vs $\Theta(\log n)$. While that is also an exponential separation, it would be consistent with, for instance, turnstile algorithms being convertible to linear sketches with an additive $O(\log n)$ loss.

Section 1.2 describes these results more formally, as well as two other similar results.

An explicit, computable reduction for deterministic algorithms. Another limitation of [LNW14], as well as the earlier work [Gan08] that applies to deterministic streaming algorithms, is that the reduction is not explicit. These reductions show the *existence* of a linear sketch, and corresponding recovery algorithm, that are equivalent to the streaming algorithm; they do not show that the sketch and recovery algorithm can be computed at all, much less computed in small space. The distinction is analogous to that of L/poly and L: they are linear sketching algorithms with a very long advice string. For an s-bit linear sketching algorithm, the advice string needs ns bits for the sketch and 2^s bits for the recovery. This is typically referred to as a "nonuniform" result, but note that the advice string is much longer than the algorithm is supposed to store: there does not necessarily exist an $O(s)$-bit machine that computes the linear sketch for each input size n and space-s streaming algorithm.

We show for deterministic streaming algorithms how to perform an *explicit* reduction: given an s-bit streaming algorithm, our algorithm computes an equivalent s-bit linear sketching algorithm in $O(s \log n)$ bits of space. To do so, we generalize what a "linear sketch" means from prior work:

Definition 1. *A* linear sketch *consists of a $\mathbb{Z}$-module homomorphism ϕ from $\mathbb{Z}^n$ to a module M.*

The "standard" linear sketch is $\phi(x) = Ax \bmod q$ for some matrix $A \in \mathbb{Z}^{m \times n}$ and set of moduli $q \in \mathbb{Z}_+^m$; the corresponding module M is $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_m}$[1]. But Definition 1 captures the ways in which standard linear sketching is useful: the sketch is linear ($\phi(x + y) = \phi(x) + \phi(y)$)), and therefore mergeable and indifferent to stream length and order.

In fact, according to the structure theorem for finitely generated $\mathbb{Z}$-modules, every linear sketch to a finite module M is equivalent to a standard linear sketch with $q_1 | q_2 | \cdots | q_m$ using the same space (i.e., $\log |M| = \log \prod q_i$). However, we do not know how to compute this transformation efficiently, and our algorithm creates a linear sketch with ϕ and M of a different form.

Theorem 2. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of all lengths, uses S space during updates and recovery, and uses s space between updates. Then there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

This reduction still has the stream length and box constraint limitations discussed in the previous section, but they are actually somewhat weaker than [LNW14, AHLW16]—the length required is exponential in s, not doubly exponential. As with these works, the above theorem applies to streaming problems representing general binary relations: any given input may have multiple valid outputs (as in approximation algorithms) or even consider every output to be valid (as in promise problems, where some inputs are invalid). For the more restrictive setting of *total functions*, where every input has a single valid output, we can remove the restriction on stream length: the same result holds for algorithms that work on streams of length $n + O(s)$.

Another advantage we believe our reduction has over prior ones is that, because it is explicit, it is easier to understand—and to understand the limitations of. We hope that this makes it easier to develop new turnstile algorithms that circumvent the limitations of these lower bounds.

We now present the definitions required to state our results more formally.

1.1 Definitions

Definition 3. *A* data stream problem *is defined by a relation $P_n \subseteq \mathbb{Z}^n \times \mathbb{Z}$. A* turnstile data stream *σ of length L is a sequence of updates $\sigma_1, \ldots, \sigma_L \in [n] \times \mathbb{Z}$. The* state *of a stream at time t is given by*

$$x^{(t)} := \operatorname{freq} \sigma^{(t)} := \sum_{(i,\Delta) \in \{\sigma_1, \ldots, \sigma_t\}} \Delta \cdot e_i.$$

and the final state is $x = \operatorname{freq} \sigma^{(L)}$.

We will also write $\operatorname{len}(\sigma)$ for L.

[1] Some descriptions of linear sketches, such as the introduction of [LNW14], omit the moduli q_i. But then the sketch would not have bounded space on all streams, so these works end up introducing moduli either explicitly (as in [LNW14]) or implicitly (by storing coordinates as $O(\log n)$-bit words with overflow). Other authors, such as [AHLW16], include the moduli.

Definition 4. *A* data stream algorithm $\mathcal{A}$ *is defined by a random distribution on initial states y; a transition function that takes a state y and a stream update σ_i and returns a new state y'; and a (possibly randomized) post-processing function g that takes the final state $\mathcal{A}(\sigma)$ and returns an output $g(\mathcal{A}(\sigma))$.*

We say that $\mathcal{A}$ solves *a problem P_n under condition C if, for all streams $\sigma \in C$, with 2/3 probability, $(\operatorname{freq}\sigma, g(y)) \in P_n$. We say that $\mathcal{A}$ uses s space between updates if all states reached by $\mathcal{A}$ while processing σ can be represented in S bits of space; we say it uses $S \geq s$ space during updates and recovery if the transition function and post-processing function use S space.*

One very common stream condition considered in the literature is that of 'strict' turnstile streams, where $x_i^{(t)} \geq 0$ for all i and t. The goal of our separations is to describe relatively mild stream conditions under which turnstile streaming is much easier than linear sketching. The goal of our equivalences is to bound S as well as s in the reduction.

For the explicit problems we consider, which are decision and counting problems, the set of valid outputs for each input forms an interval. Therefore the success probability can always be amplified to $1 - \delta$ by taking the median of $O(\log 1/\delta)$ repetitions.

Definition 5. *A* linear sketching algorithm *is a data stream algorithm where the state is $\phi(\operatorname{freq}\sigma)$, where ϕ is a linear sketch, along with the randomness used to choose ϕ.*

We will at times need some "standard" streams constructed from vectors or from other streams:

Definition 6. *For any $x \in \mathbb{Z}^n$, the "canonical" stream $\kappa(x)$ is the stream that inserts each of its coordinates in order, skipping any zero coordinates, so $\operatorname{len}(\kappa(x)) = ||x||_0$.*

For any stream σ, $\overline{\sigma}$ is the stream with the same sequence of updates but the opposite sign on each update, so if $\sigma_t = (i, \Delta)$, $\overline{\sigma}_t = (i, -\Delta)$.

For certain reductions we will need to iterate through (subsets of) $\mathbb{N}^n$ in "little-endian" order, that is, $x < y$ if $x_n < y_n$, or $x_n = y_n$ and $x_{n-1} < y_{n-1}$, and so on.

1.2 Our Results: Separations

Box-constrained streams. Our first result concerns binary streams, in which we are promised that the partial stream states $x^{(t)}$ lie in $\{0,1\}^n$ at all times.

Definition 7 (Box constraint)**.** *Γ_M is the set of streams such that for all times t, $\left\|x^{(t)}\right\|_\infty \leq M$. $\Gamma_{0,1}$ is the set of streams such that for all times t, $x^{(t)} \in \{0,1\}^n$.*

Theorem 8. *For every $n \in \mathbb{N}$, there exists a data stream problem $P_n \subseteq \{0,1\}^n \times \{0,1\}$ such that:*

1. *Any linear sketching algorithm solving P_n requires $\Omega(n^{1/3}/\log^{1/3} n)$ bits of space.*
2. *There exists a turnstile streaming algorithm that solves P_n on $\Gamma_{0,1}$ in $O(\log n)$ space.*

Note that as the final state of a linear sketching algorithm depends only on the final state of the stream, any linear sketching algorithm solving P_n on $\Gamma_{0,1}$ would also solve P_n for arbitrary streams.

One property of binary streams is that every update to a coordinate i uniquely identifies the value of x_i after the update. Over larger domains, this is no longer true. We can still show a similar result for inputs of size m, as long as intermediate results never exceed $2M - 1$:

Theorem 9. *For every $M, n \in \mathbb{N}$, there exists a data stream problem $P_n \subseteq \{-M, \ldots, M\}^n \times \{0,1\}$ such that:*

1. *Any linear sketching algorithm solving P_n requires $\Omega(n^{1/3}/\log^{1/3} n)$ bits of space.*
2. *There exists a turnstile streaming algorithm that solves P_n on Γ_{2M-1} in $O(\log n \log M)$ space.*

Interestingly, this $2M$ threshold matches one of the results in [AHLW16]. Recall that one requirement for [LNW14] to show an equivalence between linear sketching and streaming is that the streaming algorithm tolerate intermediate states of (more than) doubly exponential size, i.e., $\Gamma_{2^{2^n}}$. One result in [AHLW16] shows that this can be relaxed to Γ_{2M}—as long as $M > 2^{ns}$, where s is the algorithm space. That additional requirement is very strong (e.g., one cannot store a single coordinate of the input) but if it did not exist, the result would imply that our $2M - 1$ threshold cannot be increased.

Graph streams Our separations for binary and box-constrained streams are based on a somewhat unnatural problem. We also present separations for a more natural problem, that of counting triangles in bounded-degree graphs.

In this problem, the final state $x \in \{0,1\}^{\binom{n}{2}}$ represents a graph of maximum degree d. In the *counting* version of the problem, one would like to estimate the number of triangles T in the graph to within a multiplicative $1 \pm \varepsilon$ factor with probability $2/3$; in the *decision* version, one would like to determine whether the number of triangles is zero or at least T.

In the insertion-only model of computation, the counting problem can be solved in $O(d\frac{m}{\varepsilon^2 T}\log n)$ space [PTTW13], where $m \le nd/2$ is the number of edges in the graph, while in the linear sketching model it requires $\Omega(n/T^{2/3})$ space even for the decision version with $d = 2$ [KKP18]. This leaves a natural question: for constant d and linear T, do turnstile streaming algorithms require $\log n$ or $n^{1/3}$ space? We show, under natural conditions on the stream, that it is the former.

In our first result on this problem, we suppose that the stream represents a bounded degree graph at all times, not just at the end of the stream. In this model, we can match the best known complexity in the insertion-only model for *constant-degree* graphs [JG05, PTTW13].

Theorem 10. *There is a streaming algorithm for triangle counting in max-degree d graphs, over streams with intermediate states of max degree d, that uses $O\left(\frac{d^2 m}{\varepsilon^2 T}\log n\right)$ bits.*

When T is $\Theta(n)$, this is $O(d^3 \log n)$: exponentially smaller than the $\Omega(n^{1/3})$ lower bound for linear sketching for constant degree graphs, and still separable up to small polynomial degrees.

In our second result on this problem, we suppose that the total length of the stream is L, but allow the intermediate states to be arbitrary multigraphs.

Theorem 11. *There is a streaming algorithm for triangle counting in max-degree d graphs of length-L streams using $O\left(\frac{d^3 L^2}{\varepsilon^2 T^2}\log n\right)$ bits of space.*

For constant degree graphs with L and T both $\Theta(n)$, this is again $O(\log n)$ rather than the $\Omega(n^{1/3})$ required by linear sketching. Note that in the graph setting, n is the number of vertices (equivalently edges, as the degree is constant), and so $L = O(n)$ is equivalent to saying that at least a constant fraction of the insertions in the stream are never followed by a corresponding deletion; this is a reasonable assumption for real world graph streams such as the Facebook friends graph.

1.3 Our Results: Equivalences

Our main equivalence result is Theorem 2. We also have a slightly stronger theorem for total functions:

Theorem 12. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of length $n+2s+2$, uses S space during updates and recovery, and uses s space between updates. If P corresponds to a* total function *on $\mathbb{Z}^n$, there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

The advantage of this over Theorem 2 is that the stream length is short (i.e., $(1 + o(1))n$). The downside is that total functions are much more restrictive than binary relations, excluding approximation and promise problems.

Relative to [LNW14, AHLW16], the main benefit of Theorem 2 is that the reduction is explicitly computable in small space. The downside is that it only applies to deterministic streaming algorithms, not randomized ones. But note that even those reductions are limited in the extent to which they apply to randomized algorithms: they assume that the randomness is stored in the s space used by the algorithm. As a result, they do not apply to algorithms that flip a coin on every update, or even ones that sample a random update from the data stream: such algorithms use L and $\log L$ bits of randomness, respectively, which are much more than s for the streams considered in the reduction.

2 Related Work

Equivalences between streaming and linear sketching. As described above, [LNW14], building on [Gan08], proved that any turnstile streaming algorithm can be implemented as a linear sketch, assuming the streaming algorithm can tolerate arbitrarily long streams that feature arbitrarily complicated intermediate states. The followup work [AHLW16] removed or relaxed some of the restrictions on this equivalence: for example, they show that it still holds if the algorithm only works in the 'strict' turnstile model where all intermediate states are non-negative. They also show that it holds if the algorithm only tolerates exponentially large (in the space usage of the algorithm and the dimension of the problem) intermediate values, rather than doubly exponentially large ones.

Another line of work on the problem has considered XOR streams or other modular updates [KMY18, HLY19]. XOR streams are like binary streams, except that insert and delete updates are indistinguishable. For such streams, [HLY19] shows that for total functions the equivalence between streaming and linear sketching holds under much more mild assumptions: as long as the algorithm works on streams of length $\widetilde{O}(n^2)$. As with all the other existing equivalences, these are nonuniform: they do not show that the linear sketching algorithm is efficiently computable[2].

Lower bounds for linear sketches. The most common lower bound technique in streaming algorithms is the construction of reductions to one-way communication complexity. One encodes a hard one-way communication complexity problem into a stream by encoding Alice's input into the first half of the stream, and Bob's input into the second half. If a solution to the streaming

[2]There is some discussion in [HLY19] of generating the linear functions in small space, but this only refers to the space used to store the randomness; even in the deterministic setting, the construction is nonuniform.

problem yields a solution to the communication problem, this yields a lower bound on the streaming algorithm's space. The hard instances created by this approach tend to be fairly nice: the stream length is never more than $2n$, for example.

For linear sketching, lower bounds may also be proved by reductions to the more restrictive simultaneous message passing (SMP) model. Rather than Alice sending a short message to Bob, Alice and Bob must both send a short message to a referee, who adds their sketches to solve the problem. (One may also have more than two parties, which is typically more fruitful in the SMP model than in the one-way communication model.)

These lower bounds translate into turnstile streaming lower bounds using [LNW14, AHLW16], but the instances become horrible, leading to weak implications. In particular, this approach can never rule out algorithms using either $O(\log \log L)$ or $O(\log M)$ space, for length-L streams with intermediate states that never leave the $[-M, M]^n$ box.

Still, for a number of problems we only know how to get strong lower bounds via linear sketching. Examples include finding approximate maximum matchings [Kon15, AKLY16], estimating the size of the maximum matching [AKL17], subgraph counting [KKP18], and finding spanning forests [NY19]. Most such problems are graph problems, but the translation of the lower bound from linear sketching to streaming only applies if intermediate states are allowed to be multigraphs.

Non-linear turnstile algorithms. We are aware of one case of a turnstile streaming algorithm that is not implementable in linear sketching.

Jayaram and Woodruff [JW18] consider problems on data streams with a bounded ratio of deletions to insertions (this is similar to our condition in Theorem 11, as a long stream requires a large ratio of deletions to insertions and vice versa). The precise result depends on the problem, but roughly speaking: if the final magnitude of the vector is at least $\alpha < 1/2$ times the sum of the magnitudes of all the updates, the space complexity can be improved over linear sketches by a factor of $\log_\alpha n$. In particular, for ℓ_1 estimation, an exponential separation can be obtained, but this is $O(\log n)$ vs. $O(\log \log n)$, so even the harder case requires very little space.

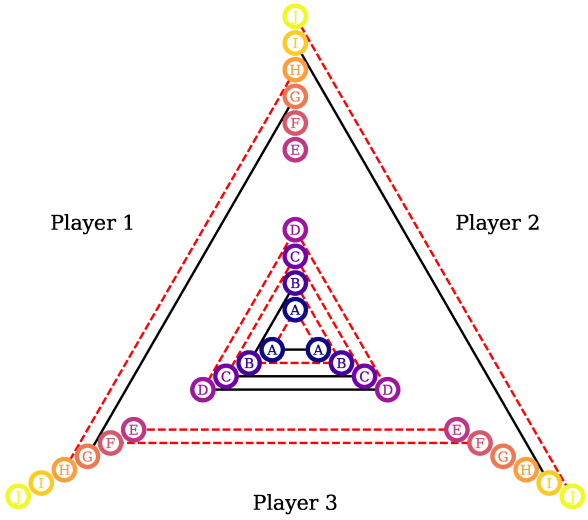
Furthermore, these results do not rule out [LNW14] being extended to short streams, as [LNW14] requires the algorithm to store all the random bits it ever uses (in contrast to the normal setting where only random bits that are to be reused have to be stored). The algorithms in [JW18] use (non-reused) randomness to sample from the updates they see, and so under this constraint they would end up needing substantially larger space. By contrast, our algorithms use only a small amount of randomness relative to their space, so they do show that a length constraint is necessary for [LNW14].

3 Overview of Techniques

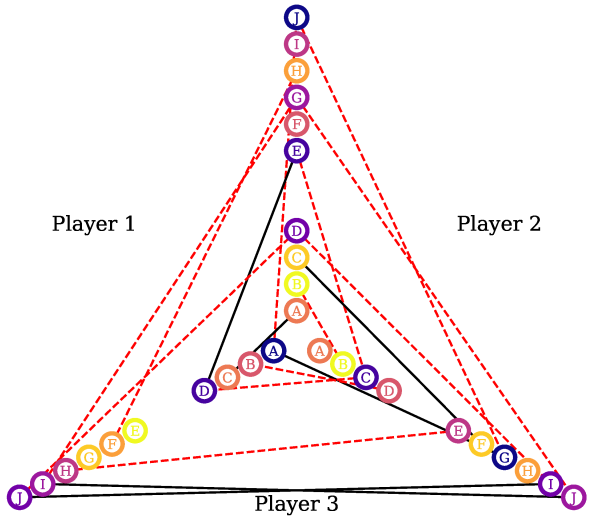
3.1 Turnstile-Sketching Separations

3.1.1 Binary and Box-Constrained Streams

Binary streams. To prove Theorem 8, we embed a hard communication problem from [KKP18] into a binary stream. In this communication problem, which we call `TrianglePromise`(n) and illustrate in Figure 1, there are three players and $O(n)$ vertices, each of which is shared between two players. Each player receives a set of $O(n)$ edges, connecting the two sets of vertices shared



(a) The player's instance ignoring the permutations. The x_e are the indices of red edges, read from inside out: $x_1 = [1, 0, 1, 1, 0, 1]$, $x_2 = [1, 1, 1, 1, 0, 1]$, $x_3 = [0, 1, 0, 0, 1, 1]$.



(b) The hard distribution permutes each set of vertices. The players see their edges and associated labels, but not the vertex colors (which represent the pre-permutation identities).

Player 1			Player 2			Player 3		
u	v	x	u	v	x	u	v	x
A	J	1	B	B	1	C	D	1
C	A	0	C	F	0	D	B	1
D	E	0	D	I	1	E	H	1
F	H	1	E	C	1	G	A	0
I	G	1	G	J	1	I	J	0
J	D	1	J	G	1	J	I	0

(c) Each player's input consists of their edges in (b). u represents the vertex counterclockwise of the player, and v represents the vertex clockwise.

	$u \to v$	$v \to u$
Player 1	$\overline{J}\perp AE\perp\overline{H}\perp\perp\overline{GD}$	$C\perp\perp\overline{J}D\perp\overline{IF}\perp\overline{A}$
Player 2	$\perp\overline{B}F\overline{IC}\perp\overline{J}\perp\perp\overline{G}$	$\perp\overline{BE}\perp\perp C\overline{J}\perp\overline{DG}$
Player 3	$\perp\perp\overline{DBH}\perp A\perp JI$	$G\overline{D}\perp\overline{C}\perp\perp\perp\overline{E}JI$

(d) The encoding into Σ^{6n}. For Theorem 8, each character in Σ is encoded into binary; for Theorem 9, the encoding is instead in $\{-m, m\}$.

Figure 1: Illustration of `TrianglePromise`(4) instance; the true instance would have 36 isolated edges per player, not 2.

with the other two players, and a label in $\{0, 1\}$ for each edge. These edges form n disjoint triangles, with each player having one edge from each triangle; every other edge is isolated. The players do not know which of their edges are in triangles. The promise is that for every triangle, the XOR of the associated bits has the same value $\tau \in \{0, 1\}$; the goal is to find τ. In [KKP18] this was shown to take $\Omega(n^{1/3})$ bits of communication in the SMP model.

Each player's input can be represented in $k = O(n \log n)$ bits. We can define a data stream problem $P \subset \{0, 1\}^{3k} \times \{0, 1\}$ as follows: for any input $x \in \{0, 1\}^{3k}$, split x into three pieces x^A, x^B, x^C, one for each player. If (x^A, x^B, x^C) represents a valid set of inputs to `TrianglePromise`(n), let τ be the corresponding answer and place (x, τ) in P; otherwise, place both $(x, 0)$ and $(x, 1)$ in P. Since the players' inputs are placed in separate coordinates, a linear sketch could solve the SMP communication problem, giving an $\Omega(n^{1/3}) = \Omega(k^{1/3}/\log^{1/3} k)$ lower bound for linear sketches. But how can we solve this problem more efficiently with an arbitrary turnstile streaming algorithm?

The lower bound in [KKP18] can be seen as proving that optimal algorithms for this problem in the SMP model must be based on *sampling*, where the players each choose a subset of their edges/bit labels to send, and succeed if there is some triangle such that each of the three players choose the edge they hold from it. What makes the problem hard, then, is the fact that it is difficult for all three players to simultaneously coordinate their sampling. Any two players can coordinate: they can use shared randomness to sample a shared vertex, and each keep their edge incident to that vertex. But they can't tell the third player which edge to keep.

The idea behind our algorithm is that for any stream, for each triangle some player's input will finish updating last. As soon as the first two players' inputs have finished updating, the algorithm will know which of their edges it sampled, and therefore know what parts of the third player's input ar. If the third player's input hasn't finished yet, the algorithm will learn at least one bit when it is updated. And to solve $\texttt{TrianglePromise}(n)$, we only need one bit.

For this to work, we need an encoding of the players' inputs that satisfies a few properties. We need to be able to sample a vertex, and learn the incident edges if we pay attention for the whole stream. If this vertex is incident to two edges of a triangle, then once we learn one of these edges, we need to know where in the vector to find the encoding of the third edge, and if we learn at least one bit of the third edge's encoding, we need to be able to be able to compute its bit label z at the end of the stream. This last point might seem tricky, but at the end of the stream the sampled edges tell us both endpoints of the third edge, so z is the only bit we don't know; it will therefore suffice to store an edge (u, v, z) as $(u \oplus z^B, v \oplus z^B)$ for a slightly larger word size B. The precise encoding and recovery algorithm are presented in Sections 4 and 4.3, respectively.

Box-constrained streams. For Theorem 9, we take the same instance as for binary streams but place it on $\{-M, M\}^{3k}$. It is no longer the case that, once we start tracking a given coordinate, we can learn its value after a single update. But we can still track the coordinate relative to its initial value, and if the coordinate's final value is M more than the smallest value seen, or M less than the largest value seen, then we *will* know the coordinate's value at the end of the stream, as there will be only one of $\{-M, M\}$ for which this is consistent with staying within Γ_{2M-1}.

Now, optimistically decoding based on the sign pattern of each word, we define the 'last' player for a triangle as being the player whose input's decoding achieves its final value last, i.e. the last player to have every coordinate of their input within $M - 1$ of its final value. At the time the first two players' inputs' decodings achieve their final value, these players will know their sampled edges, and there will be at least one coordinate of the third player's input that can be learned with the remaining stream.

3.1.2 Bounded Degree Triangle Counting

At a high level, both of our algorithms for bounded-degree triangle counting seek to emulate the insertion-only algorithm of [JG05]. The insertion-only algorithm is as follows: sample edges with probability p, and keep all edges incident to sampled edges. Count the number of triangles using sampled edges (with multiplicity if multiple edges of a triangle are sampled), and divide by p. This is an unbiased estimator, using $O(pmd \log n)$ space, in a graph with m edges, n vertices, and max degree d. The expected number of triangles sampled is pT. If all the triangles were disjoint, the triangles would be sampled independently and so one could set $p = O(1/(\varepsilon^2 T))$ and get a $(1 + \varepsilon)$-approximation with 2/3 probability. Even though the triangles are not disjoint, the degree bound keeps the estimator's variance small; one only needs $p = O(d/(\varepsilon^2 T))$.

So what happens in turnstile streams? One can run essentially the same algorithm, dealing with edge deletions by removing both the edge deleted and any neighbors that were tracked on its account. This works, but can use too much space if not done carefully.

Bounded-degree intermediate states. If every intermediate state is a bounded-degree graph, then the expected amount of space used at any point in the stream is still $O(pmd \log n)$. However, if the stream is extremely long, the *maximum* amount of space used will be too large. The natural solution is to have a hard cap of $O(pm)$ on the number of edges sampled, and to stop sampling edges when at the cap. One might worry that this creates a bias in the estimator. However, the only times this can affect the output of the algorithm are the m points in time when edges in the final graph are inserted for the last time. At each such time, with high probability, the hard cap will not have been reached. The output of the algorithm will thus be the same as in the insertion-only case.

Length-constrained streams. In this model, the intermediate states may be multigraphs with very high degree; call the maximum degree a vertex ever reaches its 'stream degree.' One cannot, in general, keep the entire neighborhood of a sampled edge. However, the $\Omega(T/d)$ edges involved in triangles in the final graph have average stream degree at most $O(\frac{Ld}{T})$. Therefore we can restrict to considering edges of stream degree $O(\frac{Ld^2}{\varepsilon T})$: this loses us at most an $\varepsilon/3d$ fraction of triangle-involved edges, which are involved in at most an ε fraction of triangles.

Using the same $p = O(d/(\varepsilon^2 T))$ as in the insertion-only case, we get an algorithm with space

$$p \cdot L \cdot \frac{Ld^2}{\varepsilon T} \cdot \log n = O(\frac{d^3 L^2}{\varepsilon^3 T^2} \log n).$$

3.2 Deterministic Turnstile-Sketching Equivalence

Our strategy for reducing deterministic turnstile streaming to linear sketching will be to take a turnstile streaming algorithm and give it various streams as input until we find vectors that can be safely "quotiented out". By repeatedly doing this we can find a linear map (a homomorphism of $\mathbb{Z}$-modules) from $\mathbb{Z}^n$ to a module of size at most 2^s, whose elements can be represented as sparse vectors in $\mathbb{Z}^n$.

In each case, the existence of these vectors will be guaranteed by the fact that $\mathcal{A}$ can have at most 2^s different states, and we will be able to find them by looking for "collisions" in these states—streams which result in different vectors but the same state of $\mathcal{A}$. How we find them, and the length of streams we will need $\mathcal{A}$ to tolerate, will depend on whether $\mathcal{A}$ calculates some total function on $\mathbb{Z}^n$ exactly, or whether it solves a general "streaming problem"—that is, each input has multiple valid outputs, e.g., a counting problem where only $(1 \pm \varepsilon)$ multiplicative accuracy is needed.

Total functions. For total functions f, we will consider streams that are the "canonical representation" $\kappa(x)$ of some vector x, defined as the stream that inserts every coordinate of x. If we can find some pair of vectors x, y such that the algorithm reaches the same state on $\kappa(x)$ and $\kappa(y)$, then for *any* vector z, the algorithm will reach the same result on $\kappa(x) \cdot \kappa(-y) \cdot \kappa(z)$ and $\kappa(y) \cdot \kappa(-y) \cdot \kappa(z)$, and so $f(z) = f(z + (x - y))$. It is therefore safe to "quotient" out $x - y$.

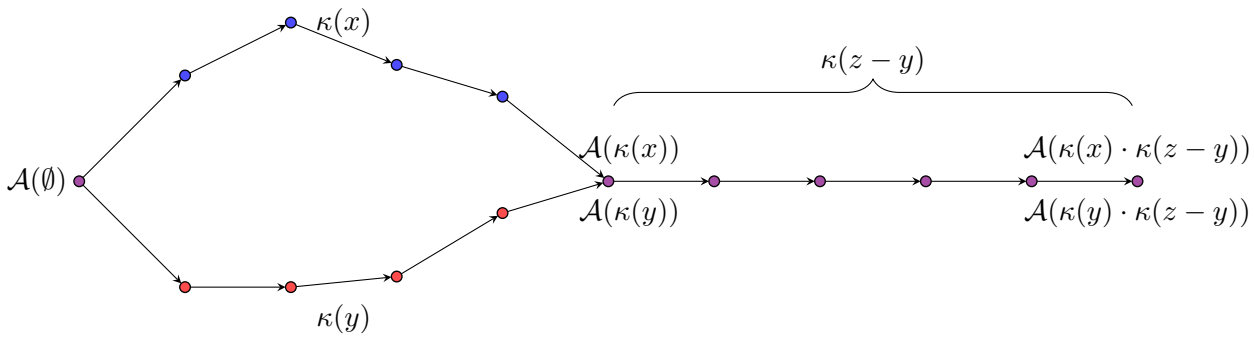


Figure 2: For total functions, we need to find pairs of streams which cause $\mathcal{A}$ to reach the same state. Here we find x and y such that their canonical representations $\kappa(x)$ and $\kappa(y)$ reach the same state. This means that for *any* z there are streams with frequency z and $z+(x-y)$ that reach the same state, so $f(z) = f(z+(x-y))$.

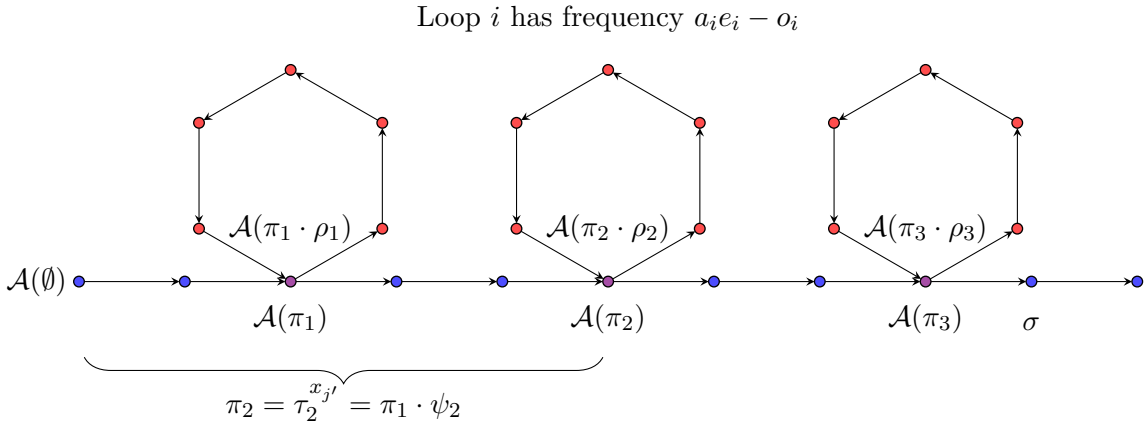


Figure 3: For general streaming problems, we generate one very long stream which iterates though a sequence of vectors x_i in $\mathbb{Z}^n$, looking for "loops" that change the value of the vector without changing the state of $\mathcal{A}$. Whatever the postfix σ is, the output will be indifferent to the number of loops ρ_i added.

By repeatedly performing this procedure, we find a submodule of $\mathbb{Z}^n$ such that f is constant on the submodule and all its cosets—our sketch can be seen as a map from $\mathbb{Z}^n$ to the corresponding quotient module.

As any vector in $\mathbb{Z}^n$ can be inserted in at most n updates, this means we only need $\mathcal{A}$ to work on length $O(n)$ streams. In fact, it will prove possible to guarantee x and y are length no more than s, so provided $\mathcal{A}$ is sublinear the required stream length is $(1 + o(1))n$.

At the end of the stream, having stored a "reduced" vector, we recover f by presenting this vector to $\mathcal{A}$ in its canonical form—as we know f takes the same value on the reduced vector as it does on the full input vector we will recover the correct answer.

General streaming problems. The above approach fails, however, if $\mathcal{A}$ has multiple valid outputs for any given input. To see this, consider the case where $\mathcal{A}$ calculates a $(1\pm\varepsilon)$ approximation to f. Then the proof above would guarantee only that $f(z)$ and $f(z + (x - y))$ were within $\varepsilon(f(z) + f(z + (x - y))$ of one another, and so repeatedly quotienting out vectors could still bring us very far from the correct answer.

So instead of finding a submodule such that f is constant on cosets of the submodule, we find a submodule such that there is a mapping from vectors in $\mathbb{Z}^n$ to streams such that for each coset of the submodule, the output of $\mathcal{A}$ on the corresponding streams is constant. We can then quotient out the vectors that generate this submodule, and then once we are finished processing the stream, map our "reduced" vector to an appropriate stream and give that stream as input to $\mathcal{A}$.

To do so we will consider a sequence of vectors x_i that iterates through $\mathbb{Z}$ in some appropriate way, and the corresponding "covering streams" $\tau^{x_i} = \kappa(x_1) \cdot \kappa(x_2 - x_1) \dots \kappa(x_i - x_{i-1})$. As $\mathcal{A}$ only has 2^s states, at some point when processing this stream it will return to a state already visited. This gives us a "loop", a sequence of updates that takes us from one state to the same state. As the x_i are distinct, we can find a loop that has non-zero frequency, and therefore we can quotient out that loop.

We repeat this process to find a sequence of streams π_i (each a prefix of the next) and loops ρ_i such that the algorithm is the same after processing $\pi_i \cdot \rho_i$ as π_i, but $\operatorname{freq} \rho_i$ is a different non-zero vector each time

For recovery, we will again insert the reduced vector in its canonical form in $\mathcal{A}$, but we will need to prefix it with the stream built up in the reduction (without loops). We then subtract off the original stream to preserve the final value of the vector. That ensures that there is some stream which corresponds to the original vector such that $\mathcal{A}$ would reach the same state it does on this one (by inserting loops[3]), and so whatever output our algorithm gives is some valid output for this vector.

Constructing a sketch. In both cases, we have described a method of finding vectors to "reduce" our input vector by—in other words, we have found a way to produce vectors that generate a submodule N of $\mathbb{Z}^n$ such that we only care which coset of N our vector is in (i.e. which element of the quotient module $\mathbb{Z}^n/N$ it maps to). However, we still need to find a consistent method to take an element x of $\mathbb{Z}^n$ to a representative element of $N + x$ that can be computed in small space.

[3]It may be noted that this will not work if taking the original vector to the reduced vector requires *subtracting* our "quotiented out" vectors. To compensate for this, our mapping from vectors to streams will include subtracting a large number of each quotient vector (outside of the loops), so that we only need to add loops. It is possible to show that there is a sufficiently large number of quotients to subtract independent of the true value of the vector.

Moreover, we need to be able, for any pair of representative elements x, y to find the representative element of $N + (x + y)$, so that we can apply module operations (i.e., maintain the sketch under updates to the stream and merge sketches of different streams).

The representative element we choose is the lexicographically first element with all non-negative coordinates in $N + x$. This can be computed in small space by repeatedly subtracting off our "quotient vectors" until it is no longer possible to do so (we will choose these vectors in a way that guarantees this eventually happens). The set of these elements will turn out to be $\prod_{i=1}^{n} \mathbb{Z}_{a_i}$ for positive integers a_i, and we will call the map from $\mathbb{Z}^n$ to this set ϕ. For any pair of representative elements x, y, the representative element of $N + (x + y)$ will be $\phi(x + y)$, so this defines a $\mathbb{Z}$-module $M \cong \mathbb{Z}^n/N$ with addition operator $\star$ given by $x \star y = \phi(x + y)$ and ϕ a homomorphism between these modules.

To actually calculate this homomorphism, we need to calculate the vectors to be quotiented out in $O(s \log n)$ space. As even storing all of them would require more space than that, we generate them sequentially whenever needed, storing only enough information about vectors generated earlier to calculate later vectors.

The proof of these results lies in Section 7.

4 Box-Constrained Streaming: Problem and lower bound

4.1 Streaming Triangle Game

Our problem is based on encoding an instance of the $\texttt{PromiseCounting}(H, n, T, \varepsilon)$ communication problem from [KKP18] as a binary vector. We will only use the special case where H is the triangle K_3, $T = n/10$, and $\varepsilon = 1$. We refer to this $\texttt{PromiseCounting}(K_3, n, n/10, 1)$ instance as $\texttt{TrianglePromise}(n)$, which we describe in Figure 4 and illustrate in Figure 1.

Theorem 13 (Implication of Corollary 15 of [KKP18])**.** *Let $n \geq 1$. Suppose that, for every instance of $\texttt{TrianglePromise}(n)$, no player sends a message of more than c bits. There exists a universal constant γ such that, if $c \leq \gamma n^{1/3}$, the probability the referee succeeds is at most 51%.*

We note that our `TrianglePromise` problem is written somewhat differently from the `PromiseCounting` problem as defined in [KKP18]. Our description is equivalent, however, as suggested in Figure 2 of [KKP18].

Both Theorem 8 and Theorem 9 involve encoding the player's inputs to $\texttt{TrianglePromise}(n)$ as a frequency vector. The outer encoding, from instances of $\texttt{TrianglePromise}(n)$ to strings from an alphabet Σ, is the same for both. The inner encoding will differ, taking strings from Σ to strings from $\{0, 1\}$ and $\{-M, M\}$ for Theorem 8 and Theorem 9 respectively.

For both, the frequency vector will have dimension $\Theta(n \log n)$. Theorems 8 and 9 then follow by considering an encoding of $\texttt{TrianglePromise}(\Theta(n/\log n))$.

`TrianglePromise` is defined in Figure 4. When there is no ambiguity about which instance of `TrianglePromise` is being referenced, we will implicitly use the variable names from this definition to refer to the corresponding variables for that instance.

Outer Encoding. We define the alphabet $\Sigma = ([N] \times \{0, 1\}) \cup \{\perp\}$. We encode an instance of $\texttt{TrianglePromise}(n)$ into Σ^{6N} as follows. For each $e \in E^{\Delta}$ and $a \in e$, we create a vector $y^{e,a} \in \Sigma^N$;

`TrianglePromise`(n)

Parties: Let V^Δ and E^Δ be the vertex and edge sets, respectively, of a triangle K_3. There are three players, one associated with each edge $e \in E^\Delta$. There is one referee, who receives messages from the three players. No other communication takes place.

Constants: Let $N = 30n$. We define N vertices V_a associated with each of the three vertices $a \in V^\Delta$.

Inputs: Each player $e = ab$ receives a list of $N/3$ triples $(u, v, z_{uv}) \in V_a \times V_b \times \{0, 1\}$.

Promise: The instance satisfies the following promise:

1. No u or v appears more than once in any single player's input. Thus the set of all edges (u, v) in player inputs can be viewed as a graph G over $\bigcup_{a \in V^\Delta} V_a$, and this graph has N edges and $3N$ vertices.
2. G contains n triangles. All $27n$ other edges are isolated.
3. There exists a $\tau \in \{0, 1\}$ such that for every triangle uvw in G,

$$z_{uv} \oplus z_{vw} \oplus z_{wu} = \tau.$$

Goal: Given the messages received from the players, the referee's task is to determine whether $\tau = 0$ or $\tau = 1$.

Figure 4: Definition of a `TrianglePromise` instance.

the full encoding is the concatenation of the six $y^{e,a}$.

As illustrated in Figure 1c, the input of player $e = ab$ consists of a list of $N/3$ edges (u, v, z_{uv}), where each $u \in V_a$ and $v \in V_b$. Since $|V_a| = |V_b| = N$, we can define a canonical bijection from each of V_a and V_b into $[N]$; call these f_a, f_b.

Then for every (u, v, z_{uv}) in player e's list, we set

$$y^{e,a}_{f_a(u)} := (f_b(v), z_{uv})$$
$$y^{e,b}_{f_b(v)} := (f_a(u), z_{uv})$$

Since each u appears at most once in e's list, this is well defined. This sets $N/3$ of the N coordinates in each of $y^{e,a}$ and $y^{e,b}$; every other coordinate is set to $\perp$.

This encoding of the players' inputs is injective; in fact, either one of $y^{e,a}$ or $y^{e,b}$ suffices to recover player e's input.

Inner Encoding. Let $B = 1 + \lceil \lg N + 1 \rceil$. For Theorem 8, we encode Σ into $\{0,1\}^B$. We encode $\perp$ as 0^B. To encode $(l, z) \in [N] \times \{0,1\}$ we first take the standard binary encoding $l^{(bin)}$ of l into $\{0,1\}^B$. This is nonzero, since $l > 0$; and its highest bit is zero, since $l \leq N$. Then we output the bitwise XOR $x = l^{(bin)} \oplus z^B$.

This encoding is injective, because the highest bit will equal z, after which z can be removed and l recovered. Concatenating the outer and inner code gives an injection from the players' inputs to $\{0,1\}^{6NB}$.

For Theorem 9, we use the same encoding, and then replace every instance of 1 with M, and every instance of 0 with $-M$.

The streaming problem. We can now define the streaming problem P_n. For any vector x such that x is *not* an encoding of an instance of $\texttt{TrianglePromise}(n)$, $(x, 0)$ and $(x, 1)$ are in P_n, i.e., any output is acceptable on such an input. For any vector x such that x is an encoding of an instance with $\tau = 0$, $(x, 0) \in P_n$, and for any vector x such that x is an encoding of an instance with $\tau = 1$, $(x, 1) \in P_n$.

4.2 Linear Sketching Lower Bound

By Theorem 13, any protocol for the communication problem that succeeds with probability at least 2/3 requires $\Omega(n^{1/3})$ bits of communication by at least one player. Furthermore, the model of [KKP18] allows the players access to an unlimited amount of shared randomness.

Now suppose we have a linear sketching algorithm for P_n. Note that the outer code encodes each player's input into separate coordinates. The inner code, of course, preserves this property. Therefore player e could encode their part of the problem with the other coordinates set to zero, sketch it, and send it to the referee. The referee can add up these sketches to get a sketch for the full vector x, then determine τ. Since each player only sends a message of size equal to the space usage of the linear sketching algorithm, the space used must be $\Omega(n^{1/3})$.

Therefore, P_n satisfies criterion 1 of Theorems 8 and 9. To prove that it satisfies criterion 2, we construct a turnstile algorithm for P_n.

4.3 Algorithm for `TrianglePromise` over $\Gamma_{0,1}$

This section will describe an algorithm that either outputs the correct answer or $\perp$, and outputs the correct answer with a small positive constant probability. Straightforward probability amplification then can increase the success probability to 2/3.

We start by noting that, for any coordinate i, we can establish x_i given any non-empty postfix of the updates to x_i, as any increase proves it was previously 0 and any decrease proves it was previously 1.

Recall that any player $e \in E^\Delta$, side $a \in e$, and vertex $u \in V_a$ has an associated symbol $y^{e,a}_{f_a(u)} \in \Sigma$. We use $x^{e,a,u} \in \{0,1\}^B$ to denote the inner encoding of this symbol. The final frequency vector x has $x^{e,a,u}$ placed in a contiguous block, at a position that is easy to find from (e, a, u).

We state the algorithm in Algorithm 1.

Algorithm 1: Low-probability `TrianglePromise` over $\{0,1\}$

1. Let (a, b, c) be a uniformly chosen random labeling of V^Δ. Choose $u \in V_a$ uniformly at random.
2. While passing through the stream:
   (a) Track all updates to $x^{ab,a,u}$ and $x^{ac,a,u}$.
   (b) While doing so, keep checking whether $x^{ab,a,u}$ is a valid inner encoding of Σ; if it is, and it doesn't decode to $\perp$, then it is an encoding of $(f_b(v'), z)$ for some $v' \in V_b$ and z'. Let (v', z') be those values, if they exist.
   (c) As soon as (v', z') is set, track all updates to $x^{bc,b,v'}$. Discard these updates whenever (v', z) changes.
3. After the stream finishes:
   (a) Decode $x^{ab,a,u}$ and $x^{ac,a,u}$ to Σ.
   (b) If either is $\perp$, output $\perp$.
   (c) Otherwise, let their decodings be $(f_b(v), z_{uv})$ and $(f_c(w), z_{uw})$ for $v \in V_b$ and $w \in V_c$.
   (d) If the algorithm has not tracked any updates to $x^{bc,b,v}$, output $\perp$.
   (e) Otherwise, it knows $x^{bc,b,v}_i$ for some index $i \in [B]$. Let $z_{vw} = x^{bc,b,v}_i \oplus f_c(w)^{(bin)}_i$.
   (f) Output $z_{uv} \oplus z_{vw} \oplus z_{uw}$.

Lemma 14. *The space complexity of Algorithm 1 is $O(\log n)$ bits.*

Proof. The randomness in step 1 uses $\log(6N)$ bits. After that, the algorithm tracks three length-B vectors; the total space usage is $O(\log n)$. □

Lemma 15. *Algorithm 1 outputs either $\perp$ or τ. If u is part of a triangle in the underlying `TrianglePromise`(n) graph G, and the last stream update to $x^{ab,a,u}$ is before the last stream update to $x^{bc,b,v}$, then the algorithm outputs τ.*

Proof. Note that $x^{ab,a,u}$ and $x^{ac,a,u}$ are tracked completely, so their final decodings into Σ are correct. If u is not part of a triangle, at most one edge is incident to u in the full graph G, so at least one of the decodings is $\perp$ and the algorithm returns $\perp$.

Otherwise, if u is part of a triangle, the algorithm correctly deduces (v, z_{uv}) and (w, z_{uw}). If the algorithm has not seen an update to $x^{bc,b,v}$, it will output $\perp$; otherwise, since it tracks a postfix of the stream, it correctly identifies $x_i^{bc,b,v}$. Since uvw is a triangle, we know player bc has the input (v, w, z_{vw}) for some vw, and the inner encoding is

$$x_i^{bc,b,v} = z_{vw} \oplus f_c(w)_i^{(bin)}.$$

Thus the algorithm correctly identifies z_{vw}, and the $\texttt{TrianglePromise}(n)$ promise says

$$\tau = z_{uv} \oplus z_{vw} \oplus z_{uw}.$$

Hence the algorithm outputs either $\perp$ or τ. Moreover, it will have deduced v correctly upon the last update to $x^{ab,a,u}$; if this is before the last update to $x^{bc,b,v}$ then it will see at least one update there and output τ. □

Lemma 16. *Algorithm 1 outputs τ with at least $\frac{1}{180}$ probability.*

Proof. There is a $n/N = 1/30$ chance that u lies in a triangle, independent of the choice of (a, b, c). Furthermore, if it does, *which* triangle it lies in is independent of the choice of (a, b, c).

Suppose u lies in the triangle uvw with $u \in V_{a'}, v \in V_{b'}, w \in V_{c'}$. One of the three blocks

$$x^{a'b',a',u}, \qquad x^{b'c',b',v}, \qquad x^{c'a',c',w}$$

will be the first to finish being updated in the stream. WLOG this is a'. Then Lemma 15 says that if $(a, b, c) = (a', b', c')$, Algorithm 1 will output τ. This choice happens with $1/6$ probability; combined with the $1/30$ chance that u lies in a triangle, we get at least a $1/180$ chance of outputting τ. □

Lemma 17. *There is a turnstile streaming algorithm that solves P_n on $\Gamma_{0,1}$ with probability $2/3$ using $O(\log n)$ bits of space.*

Proof. Run Algorithm 1 in parallel 360 times and output any non-$\perp$ result. By Lemma 15 any non-$\perp$ result will be correct. By Lemma 16 the failure probability is at most $(1 - 1/180)^{360} < 1/e^2 < 1/3$. □

4.4 Algorithm for $\texttt{TrianglePromise}$ over Γ_{2M-1}

We write $\sigma^{(t)}$ for the prefix of σ consisting of its first t updates. Define the error correction function ζ by

$$\zeta(z)_i = \begin{cases} M & z_i > 0 \\ -M & z_i < 0 \\ 0 & z_i = 0 \end{cases}$$

and define the decoding function $\eta : \{-M, M\}^* \to \{0, 1\}$ by:

$$\eta(z)_i = \begin{cases} 1 & z_i = M \\ 0 & z_i = -M \end{cases}$$

We will use the following decoding lemma in our algorithm:

Lemma 18. *Let σ be a stream in Γ_{2M-1} such that $\operatorname{freq}\sigma \in \{-M, M\}^*$. Then for any i, and for any split of the stream $\sigma = \sigma_1 \cdot \sigma_2$,*

1. $\min_t(\operatorname{freq}\sigma_2^{(t)})_i \le (\operatorname{freq}\sigma_2)_i - M \Rightarrow \eta(\operatorname{freq}\sigma)_i = 1$
2. $\max_t(\operatorname{freq}\sigma_2^{(t)})_i \ge (\operatorname{freq}\sigma_2)_i + M \Rightarrow \eta(\operatorname{freq}\sigma)_i = 0$

and one of these conditions holds iff $\exists t$ such that $\zeta(\operatorname{freq}\sigma_1 \cdot \sigma_2^{(t)})_i \neq \zeta(\operatorname{freq}\sigma)_i$.

Proof. Suppose $\min_t(\operatorname{freq}\sigma_2^{(t)})_i \le (\operatorname{freq}\sigma_2)_i - M$. Then if $\eta(\operatorname{freq}\sigma)_i = 0$, $(\operatorname{freq}\sigma)_i = -M$. Let t be a minimizer of $(\operatorname{freq}\sigma_2^{(t)})_i$, so

$$\begin{aligned}(\operatorname{freq}\sigma^{(|\sigma_1|+t)})_i &= (\operatorname{freq}\sigma_1)_i + (\operatorname{freq}\sigma_2^{(t)})_i \\ &\le (\operatorname{freq}\sigma_1)_i + (\operatorname{freq}\sigma_2)_i - M \\ &= (\operatorname{freq}\sigma)_i - M \\ &= -2M\end{aligned}$$

but by the box constraint $(\operatorname{freq}\sigma^{(t)})_i \ge -2M + 1$, giving a contradiction. So $\eta(\operatorname{freq}\sigma)_i = 1$.

Likewise, if $\max_t(\operatorname{freq}\sigma_2^{(t)})_i \ge (\operatorname{freq}\sigma_2)_i + M$, there exists t such that if $\eta(\operatorname{freq}\sigma)_i = 1$, $(\operatorname{freq}\sigma^{(|\sigma_1|+t)})_i \ge 2M$, so it must be the case that $\eta(\operatorname{freq}\sigma)_i = 0$.

For the final part of the lemma, note that one of the conditions holds iff

$$\max_t |(\operatorname{freq}\sigma_2^{(t)})_i - (\operatorname{freq}\sigma_2)_i| \ge M$$

or equivalently iff

$$\max_{t \ge |\sigma_1|} |(\operatorname{freq}\sigma^{(t)})_i - (\operatorname{freq}\sigma)_i| \ge M$$

which as $(\operatorname{freq}\sigma)_i = \pm M$, holds iff there is a $t \ge |\sigma_1|$ such that either $(\operatorname{freq}\sigma^{(t)})_i \le 0$ and $(\operatorname{freq}\sigma)_i = M$, or $(\operatorname{freq}\sigma^{(t)})_i \ge 0$ and $(\operatorname{freq}\sigma)_i = -M$, and in turn one of these holds iff $\zeta(\operatorname{freq}\sigma^{(t)}) \neq \zeta(\operatorname{freq}\sigma)$. □

The algorithm is described in Algorithm 2.

Lemma 19. *The space complexity of Algorithm 2 is $O(\log n \log M)$ bits.*

Proof. The randomness in step 1 uses $\log(6N)$ bits. After that, the algorithm tracks three length-B vectors with entries in $\{-M, M\}$; the total space usage is $O(\log n \log M)$. □

Lemma 20. *Algorithm 2 outputs either $\perp$ or τ. If u is part of a triangle in the underlying* `TrianglePromise`(n) *graph G, and the last time $\zeta(x^{ab,a,u})$ differs from its final value is before the last time $\zeta(x^{bc,b,v})$ differs from its final value, then the algorithm outputs τ.*

Proof. Note that $x^{ab,a,u}$ and $x^{ac,a,u}$ are tracked completely, so their final decodings into Σ are correct. If u is not part of a triangle, at most one edge is incident to u in the full graph G, so at least one of the decodings is $\perp$ and the algorithm returns $\perp$.

Otherwise, if u is part of a triangle, the algorithm correctly deduces (v, z_{uv}) and (w, z_{uw}). If the last time $\zeta(x^{ab,a,u})$ differs from its final value is *after* the last time $\zeta(x^{bc,b,v})$ differs from its final

Algorithm 2: Low-probability `TrianglePromise` over Γ_{2M-1}

1. Let (a, b, c) be a uniformly chosen random labeling of V^{Δ}. Choose $u \in V_a$ uniformly at random.
2. While passing through the stream:
   (a) Track all updates to $x^{ab,a,u}$ and $x^{ac,a,u}$.
   (b) While doing so, keep checking whether $\zeta(x^{ab,a,u})$ is a valid inner encoding of Σ; if it is, and it doesn't decode to $\perp$, then it is an encoding of $(f_b(v'), z)$ for some $v' \in V_b$ and z'. Let (v', z') be those values, if they exist.
   (c) As soon as (v', z') is set, track all updates to $x^{bc,b,v'}$, recording the current, minimum, and maximum value of each of its coordinates. Discard these updates whenever (v', z) changes.
3. After the stream finishes:
   (a) Decode $\zeta(x^{ab,a,u})$ and $\zeta(x^{ac,a,u})$ to Σ.
   (b) If either is $\perp$, output $\perp$.
   (c) Otherwise, let their decodings be $(f_b(v), z_{uv})$ and $(f_c(w), z_{uw})$ for $v \in V_b$ and $w \in V_c$.
   (d) If the final observed value for $x^{bc,b,v}$ is within $M-1$ of all the values the algorithm has observed for it, output $\perp$.
   (e) Otherwise, by Lemma 18 it knows $\eta(x^{bc,b,v})_i$ for some index $i \in [B]$. Let $z_{vw} = \eta(x^{bc,b,v})_i \oplus f_c(w)_i^{(bin)}$.
   (f) Output $z_{uv} \oplus z_{vw} \oplus z_{uw}$.

value, then at the time the algorithm starts tracking $x^{bc,b,v}$, $\zeta(x^{bc,b,v})$ has already its final value, and so by Lemma 18, the final observed value for $x^{bc,b,v}$ is within $M-1$ of all the values observed for it, and so the algorithm outputs $\perp$. Otherwise, by Lemma 18, the algorithm correctly identifies $\eta(x^{bc,b,v})_i$.

Since uvw is a triangle, we know player bc has the input (v, w, z_{vw}) for some vw, and we know

$$\eta(x^{bc,b,v})_i = z_{vw} \oplus f_c(w)_i^{(bin)}.$$

Thus the algorithm correctly identifies z_{vw}, and the `TrianglePromise`(n) promise says

$$\tau = z_{uv} \oplus z_{vw} \oplus z_{uw}.$$

Hence the algorithm outputs either $\perp$ or τ, and the last time $\zeta(x^{ab,a,u})$ differs from its final value is before the last time $\zeta(x^{bc,b,v})$ differs from its final value, then the algorithm outputs τ. □

Lemma 21. *Algorithm 2 outputs τ with at least $\frac{1}{180}$ probability.*

Proof. There is a $n/N = 1/30$ chance that u lies in a triangle, independent of the choice of (a, b, c). Furthermore, if it does, *which* triangle it lies in is independent of the choice of (a, b, c).

Suppose u lies in the triangle uvw with $u \in V_{a'}, v \in V_{b'}, w \in V_{c'}$. WLOG, let $\zeta(x^{a'b',a',u})$ stop changing before $\zeta(x^{b'c',b',v})$ or $\zeta(x^{c'a',c',w})$.

Then Lemma 20 says that if $(a, b, c) = (a', b', c')$, Algorithm 2 will output τ. This choice happens with $1/6$ probability; combined with the $1/30$ chance that u lies in a triangle, we get at least a $1/180$ chance of outputting τ. □

Lemma 22. *There is a turnstile streaming algorithm that solves P_n on Γ_{2M-1} with probability $2/3$ using $O(\log n \log M)$ bits of space.*

Proof. Run Algorithm 2 in parallel 360 times and output any non-$\perp$ result. By Lemma 20 any non-$\perp$ result will be correct. By Lemma 21 the failure probability is at most $(1 - 1/180)^{360} < 1/e^2 < 1/3$. □

5 Restricted Intermediate State Triangle Counting

5.1 Problem

Valid inputs to our problem will be as follows (for invalid inputs, any output is accepted): x will be a binary string indexed by $E(K_n)$, the set of all possible edges on an n-vertex graph. We will associate it with a graph G on n vertices with edge set $\{e \in E(K_n) : x_e = 1\}$. We will use m to denote the size of this edge set. Finally, G has max degree d.

Instead of bounding the length of the stream, we will require that $x^{(t)}$ correspond to a graph G with max degree d for *all* t. One consequence of this is that all updates will be in $[-1, 1]$.

Our problem will be to estimate T, the number of triangles in the graph, up to some multiplicative precision ε. Our algorithm will succeed in doing this if the space allocated to it is large enough in terms of T. This space requirement is decreasing in T, so we may express this as a data stream problem in the sense of Definition 3 by choosing a lower bound T' and making any answer acceptable for an input vector x that does not correspond to a valid input or results in $T < T'$, and

making all outputs in $[(1-\varepsilon)T, (1+\varepsilon)T]$ acceptable for input vectors that correspond to a valid graph with $T \geq T'$.

5.2 Linear Sketching Lower Bound

By Theorem 7 of [KKP18], any sketching algorithm for this problem requires $\Omega(m/T^{1/3})$ bits. The requirement that d be constant does not affect this, as the [KKP18] reduction is on graphs of max degree 2. Neither does the intermediate state requirement, as the output of a sketching algorithm depends only on the final state of the stream.

5.3 Algorithm

1. Initialize our set of seed edges $S = \emptyset$. Let $h : E \to \{0, 1\}$ be a threewise independent hash function where $h(e) = 1$ with probability p.
2. While passing through the stream:
   (a) On receiving an update $(e, +1)$:
   - If $h(e) = 1$ and $|S| \leq 2pm$, add e to S, and initialize S_e as $\emptyset$.
   - If $\exists f \in S$ such that e is incident to f, add e to S_f.

   (b) On receiving an update $(e, -1)$:
   - Remove it from any of S and the sets S_f that contain it.
   - Delete the set S_e if it exists.
3. For each $e = uv$, set
$$\widetilde{T}_e = \begin{cases} p^{-1}|\{w : uw, vw \in S_e\}| & \text{if } e \in S \\ 0 & \text{otherwise.} \end{cases}$$
4. Return $\widetilde{T} = \sum_e \widetilde{T}_e$.

5.4 Space Complexity

Lemma 23. *This algorithm requires $O(pdm \log n)$ bits of space.*

Proof. The set S has size at most $2pm$ at any point in time, and for each element e in S at most $2d-1$ edges are kept (as each endpoint of e has degree at most d at all times), and each edge takes $O(\log n)$ bits of space to store. □

5.5 Correctness

Definition 24. *$G^{(t)}$ and $S^{(t)}$ denote the state of G and S respectively after the first t updates, so that $G^{(L)} = G$ and $S^{(L)} = S$.*

Definition 25. *For any edge $e \in G$, let t_e denote the time of the last update made to e. For any triangle $\tau \in G$, let $\rho(\tau)$ denote the edge $e \in \tau$ that minimizes t_e. Then:*

$$T_e = |\{\tau : \rho(\tau) = e\}|$$

Note that as each triangle τ has exactly one e such that $\rho(\tau) = e$, $T = \sum_e T_e$.

Definition 26. *Let $Q^{(t)} = \{e \in E(G^{(t)}) : h(e) = 1\}$, $Q = Q^{(L)}$, and $Q_e = \{f$ incident to $e : t_f > t_e\}$. Then:*

$$\widetilde{T}_e^+ = \begin{cases} p^{-1}|\{w : uw, vw \in Q_e\}| & \text{if } e \in Q \\ 0 & \text{otherwise.} \end{cases}$$
$$\widetilde{T}^+ = \sum_e \widetilde{T}_e^+$$

Lemma 27.

$$\mathbb{E}\left[\widetilde{T}^+\right] = T$$
$$Var(\widetilde{T}^+) \leq p^{-1}dT$$

Proof. For each $e \in E(G)$, $\widetilde{T}_e^+ = p^{-1}T_e$ if $h(e) = 1$ and 0 otherwise. So

$$\mathbb{E}\left[\widetilde{T}_e^+\right] = T_e$$
$$\begin{aligned} \mathrm{Var}(\widetilde{T}_e^+) &\leq T_e^2/p \\ &\leq dT_e/p \end{aligned}$$

and as h is threewise independent:

$$\begin{aligned} \mathbb{E}\left[\widetilde{T}^+\right] &= \sum_e T_e \\ &= T \\ \mathrm{Var}(\widetilde{T}^+) &= \sum_e \mathrm{Var}(\widetilde{T}_e^+) \\ &\leq dT/p. \end{aligned}$$

□

Lemma 28. *For any $e \in Q$, if $|S^{(t_e-1)}| < 2pm$, $\widetilde{T}_e = \widetilde{T}_e^+$. Otherwise, $\widetilde{T}_e = 0$.*

Proof. If $e \in Q$, it will be in S unless S is size $2pm$ at the final time it would be added (if it is added earlier, it will be deleted before time t_e, so only the size of $S^{(t_e)}$ matters). Furthermore, if it is added, the edges in S_e will be precisely those edges of G that have their final update after S_e is created for the last time, that is, after t_e. So if $|S^{(t_e-1)}| < 2pm$, $\widetilde{T}_e = \widetilde{T}_e^+$.

On the other hand, if $|S^{(t_e-1)}| = 2pm$, then $e \notin S^{(t_e-1)}$, as it will have been deleted since the last time it might have been added, $e \notin S^{(t_e)}$, as it will not be added, and so $e \notin S$, as there are no more updates to e. □

Lemma 29. *For all $e \in E(G)$:*

$$\mathbb{P}\left[|S^{(t_e-1)}| = 2pm \middle| h(e) = 1\right] \leq 1/pm$$

Proof. By the intermediate state condition on $G^{(t_e-1)}$, it has at most m edges. Then as $S^{(t_e-1)} \subseteq Q^{(t_e-1)}$, and as h is threewise independent and $h(e) = 1$ with probability p,

$$\mathbb{E}\left[|Q^{(t_e-1)}|\middle|h(e)=1\right] \leq pm$$
$$\mathrm{Var}\left(|Q^{(t_e-1)}|\middle|h(e)=1\right) \leq (p-p^2)m$$

so by Chebyshev's inequality:

$$\begin{aligned}\mathbb{P}\left[|S^{(t_e-1)}| = 2pm\middle|h(e)=1\right] &\leq \mathbb{P}\left[|Q^{(t_e-1)}| \geq 2pm\middle|h(e)=1\right]| \\ &\leq 1/pm\end{aligned}$$

□

Lemma 30.

$$\mathbb{E}\left[|\widetilde{T} - \widetilde{T}^+|\right] \leq T/pm$$

Proof. By Lemma 28, $|\widetilde{T}_e - \widetilde{T}_e^+| = p^{-1}T_e$ if $h(e) = 1$ and $|S^{(t_e-1)}| = 2pm$, and 0 otherwise. So, Lemma 29:

$$\begin{aligned}\mathbb{E}\left[|\widetilde{T} - \widetilde{T}^+|\right] &\leq \sum_e \mathbb{E}\left[|\widetilde{T}_e - \widetilde{T}_e^+|\right] \\ &\leq \sum_e p^{-1}T_e\,\mathbb{P}\left[|S^{(t_e-1)}| = 2pm \wedge h(e)=1\right] \\ &\leq \sum_e (T_e/p^2m)\,\mathbb{P}\left[h(e)=1\right] \\ &= T/pm\end{aligned}$$

□

Theorem 10. *There is a streaming algorithm for triangle counting in max-degree d graphs, over streams with intermediate states of max degree d, that uses $O\left(\frac{d^2m}{\varepsilon^2 T}\log n\right)$ bits.*

Proof. Let the algorithm be run with $p = 32d/\varepsilon^2 T$. Then by Lemma 30,

$$\begin{aligned}\mathbb{E}\left[|\widetilde{T} - \widetilde{T}^+|\right] &\leq T^2/32dm \\ &\leq T/32 \qquad\qquad \text{as } T \leq dm.\end{aligned}$$

Therefore, by Markov's inequality:

$$\mathbb{P}\left[|\widetilde{T} - \widetilde{T}^+| \geq \varepsilon T/2\right] \leq 1/16$$

Then, by Lemma 27,

$$\mathbb{E}\left[\widetilde{T}^+\right] = T$$
$$\mathrm{Var}(\widetilde{T}^+) \leq T^2/8$$

and so by Chebyshev's inequality,

$$\mathbb{P}\left[|\widetilde{T}^{+} - T| \geq \varepsilon T/2\right] \leq 1/4$$

so:

$$\mathbb{P}\left[|\widetilde{T} - T| \geq \varepsilon T\right] \leq 5/16$$

Therefore, by running $O(\log 1/\delta)$ copies of the algorithm in parallel and taking the median, we can output a $(1 \pm \varepsilon)$ multiplicative approximation to T with probability $1 - \delta$. □

6 Bounded-Length Triangle Counting

6.1 Problem

We will work in the *strict* turnstile model, so our input vector $x = \operatorname{freq} \sigma^{(L)}$ is non-negative at all intermediate steps.

Valid inputs to our problem will be as follows (for invalid inputs, any output is accepted): x will be indexed by $E(K_n)$, the set of all possible edges on an n-vertex graph. We will associate it with a graph G on n vertices with edge set $\{e \in E(K_n) : x_e = 1\}$. x is binary, but its intermediate states may not be. We will use m to denote the size of this edge set. Finally, G has max degree d.

Our problem will be to estimate T, the number of triangles in the graph, up to some multiplicative precision ε. Our algorithm will succeed in doing this if the space allocated to it is large enough in terms of T. This space requirement is decreasing in T, so we may express this as a data stream problem in the sense of Definition 3 by choosing a lower bound T' and making any answer acceptable for an input vector x that does not correspond to a valid input or results in $T < T'$, and making all outputs in $[(1-\varepsilon)T, (1+\varepsilon)T]$ acceptable for input vectors that correspond to a valid graph with $T \geq T'$.

6.2 Linear Sketching Lower Bound

By Theorem 7 of [KKP18], any sketching algorithm for this problem requires $\Omega(m/T^{1/3})$ bits. The requirement that d be constant does not affect this, as the [KKP18] reduction is on graphs of max degree 2, and neither do the stream length and strict turnstile requirements, as they will not affect the output of any linear sketch.

6.3 Algorithm

1. Initialize our set of seed edges $S = \emptyset$. Let $h : E \to \{0, 1\}$ be a pairwise independent hash function where $h(e) = 1$ with probability p.
2. While passing through the stream, on receiving an update (e, χ):
   - If $h(e) = 1$, and there is no tuple $(e, \gamma) \in S$, add (e, χ) to S.
   - If $h(e) = 1$, and $(e, \gamma) \in S$, replace it with $(e, \chi + \gamma)$.
   - If (e, χ) has been added to S for some $\chi > 0$, initialize the set $S_e = \emptyset$.
   - If $(e, 0)$ is now in S, delete S_e.

- Then, for each f incident to e such that $(f, z) \in S$:
  - If $(e, \gamma) \in S_f$, replace it with $(e, \max(\chi + \gamma, 0))$.
  - Otherwise, insert $(e, \max(\chi, 0))$ into S_f, unless $|S_f| \geq \frac{2d^2L}{\varepsilon T}$.

3. For each edge $e = uv$, set:

$$\widetilde{T}_e = \begin{cases} p^{-1}|\{w : (uw, 1), (vw, 1) \in S_e\}| & \text{If } (e, 1) \in S. \\ 0 & \text{Otherwise.} \end{cases}$$

4. Return $\widetilde{T} = \sum_e \widetilde{T}_e$.

6.4 Space Complexity

Lemma 31. *The expected space complexity of this algorithm is at most* $O\left(\frac{pd^2L^2}{\varepsilon T} \log n\right)$ *bits.*

Proof. Each edge in the stream is independently included in S with probability p, so the expected maximum size of S is at most pL. For each element of S we keep an integer of size $\text{poly}(n)$, requiring $O(\log n)$ bits, and a set of size no more than $\frac{2d^2L}{\varepsilon T}$. The elements of these sets are edges of an n-vertex graph, and integers of size $\text{poly}(n)$, and therefore require $O(\log n)$ bits each to represent. □

6.5 Correctness

Consider some fixed (strict) turnstile stream of length L. Let G be the graph with vertex set $[n]$ and edge set $\{e \in E : x_e = 1\}$, and let T be the number of triangles in G. We will seek to show that this algorithm can approximate T.

Definition 32. *For any edge $e \in G$, let t_e be the largest $t \in [L]$ such that:*

$$x_e^{(t-1)} = 0$$
$$x_e^{(t)} > 0$$

For any triangle $\tau \in G$, let $\rho(\tau) \in \tau$ be the edge of τ that maximizes $t_{\rho(\tau)}$. Then, define:

$$T_e = |\{\tau \in G : \rho(\tau) = e\}|$$

Note that as each triangle τ has exactly one edge e such that $\rho(\tau) = e$, $\sum_e T_e = T$.

Definition 33. *For any edge $e \in G$ and $t \geq t_e$, $Q_e^{(t)}$ is the set generated by the following procedure:*

- *For $t' = t_e, \ldots, t$, and $(f, \chi) = \sigma_{t'}$, if f is incident to e:*
  - *If $(f, \gamma) \in Q_e^{(t)}$, replace it with $(e, \max(\chi + \gamma, 0))$.*
  - *Otherwise, insert $(f, \max(\chi, 0))$ into $Q_e^{(t)}$.*

Lemma 34. *For any e such that $h(e) = 1$,*

$$Q_e^{(L)} \supseteq S_e$$

with equality when

$$|Q_e^{(L)}| \leq \frac{2d^2L}{\varepsilon}.$$

Proof. As $h(e) = 1$, S_e will be deleted and recreated for the final time at t_e. After this point, the procedures for creating S_e and $Q_e^{(L)}$ are identical as long as $|S_e|$ (and therefore $Q_e^{(L)}$) never reaches size $\frac{2d^2L}{\varepsilon}$. If it does, the only difference is that some edges may be excluded from S_e. □

For any (f, z) such that $f \in Q_e^{(t)}$ we will also write $f \in Q_e^{(t)}$, and $Q_e^{(t)}[f] = z$. Note that $Q_e^{(r)}[f]$ is well-defined whenever $f \in Q_e^{(t)}$ (as no edge is added to $Q_e^{(t)}$ more than once) and $f \in Q_e^{(t)} \Rightarrow f \in Q_e^{(t+1)}$ (as no edges are ever removed from $Q_e^{(r)}$.

Lemma 35. *For all edges f incident to e and integers $t \in [t_e, L]$,*

$$Q_e^{(t)}[f] = x_f^{(t)} - \min_{r=t_e,\dots,t} x_f^{(r)}$$

Proof. We proceed by induction on t. If $t = t_e$, as the update at time t_e was to e, $Q_e^{(t)}[f] = 0$ and so the result holds. Now suppose $t > t_e$ and $Q_e^{(t-1)}[f] = x_f^{(t-1)} - \min_{r=t_e,\dots,t-1} x_f^{(r)}$.

Then, let $\sigma_t = (f', \chi)$. If $f' \neq f$ both sides of the equation are unchanged and we are done. So suppose the update is (t, f, χ). We will consider two cases.

$\mathbf{Q_e^{(t-1)}[f] + \chi \geq 0}$ Then $Q_e^{(t)} = Q_e^{(t-1)}[f] + \chi$ and $x_f^{(t)} = x_f^{(t-1)} + \chi$. Furthermore, $\chi \geq -Q_e^{(t-1)}[f]$, so we have:

$$\begin{aligned} x_f^{(t)} &= x_f^{(t-1)} + \chi \\ &\geq x_f^{(t-1)} - Q_e^{(t-1)}[f] \\ &= \min_{r=t_e,\dots,t-1} x_f^{(r)} \end{aligned}$$

So $\min_{r=t_e,\dots,t} x_f^{(r)} = \min_{r=t_e,\dots,t-1} x_f^{(r)}$, completing the proof.

$\mathbf{Q_e^{(t-1)}[f] + \chi < 0}$ Then $Q_e^{(t)} = 0$, and:

$$\begin{aligned} x_f^{(t)} &= x_f^{(t-1)} + \chi \\ &< x_f^{(t-1)} - Q_e^{(t-1)} \\ &= \min_{r=t_e,\dots,t-1} x_f^{(r)} \end{aligned}$$

So $\min_{r=t_e,\dots,t} x_f^{(r)} = x_f^{(t)}$, and so $x_f^{(t)} - \min_{r=t_e,\dots,t} x_f^{(r)} = 0$, completing the proof.

□

Definition 36. *For any vertex x, let the 'stream degree' l_v be the number of edges e incident to x such that there is some update $\sigma_t = (e, \chi)$, regardless of whether e is in the final graph G.*

Lemma 37. *Let $e = uv$ be an edge. Then*

$$\widetilde{T}_e = \begin{cases} \overline{T}_e/p & \text{with probability } p \\ 0 & \text{otherwise.} \end{cases}$$

where $\overline{T}_e = T_e$ if $l_u + l_v \leq \frac{2d^2L}{\varepsilon T}$, and $\overline{T}_e \in [0, T_e]$ otherwise.

Proof. Let e be an edge. If $h(e) = 0$, $(e, 1) \notin S$, and so $\widetilde{T}_e = 0$. This event happens with probability $1 - p$. If $h(e) = 1$ but $e \notin G$, $x_e = 0$, and so $(e, 1) \notin S$, so $\widetilde{T}_e = 0 = T_e = \widetilde{T}_e$.

Now consider the case where $h(e) = 1$ and e in G. Then $x_e = 1$, so $(e, 1) \in S$. $\widetilde{T}_e$ will then be p^{-1} times the number of triangles uvw, where $e = uv$ and $(uw, 1), (vw, 1) \in S_e$. If $l_u + l_v \leq \frac{2d^2L}{\varepsilon T}$, then $|Q_e^{(L)}| \leq \frac{2d^2L}{\varepsilon T}$ and so by Lemma 34, $Q_e^{(L)} = S_e$, and otherwise $Q_e^{(L)} \supseteq S_e$.

So it will suffice to show that

$$|\{w : (uw, 1), (vw, 1) \in Q_e^{(L)}\}| = |\{\tau \in G : \rho(\tau) = e\}|$$

. We will show that

$$\{f : (f, 1) \in Q_e^{(L)}\} = \{f \in G : t_f > t_e, \, e \text{ incident to } f\}$$

which implies our result, as it means that $w \in \{w : (uw, 1), (vw, 1) \in Q_e^{(L)}\}$ iff the triangle uvw has $t_{uv} < t_{uw}, t_{vw}$.

For any $f \in E$ incident to e, by Lemma 35, $(f, 1) \in Q_e^{(L)}$ iff $x_f^{(L)} - \min_{r=t_e,\ldots,L} x_f^{(r)} = 1$. If $f \notin G$, then $x_f^{(L)} = 0$ and so this cannot hold, as $x_f^{(r)} \geq 0$ for all r. If $f \in G$, then $x_f^{(L)} = 1$ and so this holds iff $\min_{r=t_e,\ldots,L} x_f^{(r)} = 0$, that is, iff $t_f > t_e$. So $(f, 1) \in Q_e^{(L)}$ iff $f \in G$ and $t_f > t_e$, concluding the proof. □

Lemma 38.

$$\mathbb{E}\left[\widetilde{T}\right] \in [(1 - \varepsilon/2)T, T]$$

Proof. By Lemma 37, $\mathbb{E}\left[\widetilde{T}\right] = \sum_e \overline{T}_e$, where $\overline{T}_e = T_e$ if $l_u + l_v \leq \frac{2d^2L}{\varepsilon T}$ and $\overline{T}_e \in [0, T_e]$ otherwise. Recalling that $T_e = |\{\tau \in G : \rho(\tau) = e\}|$, this gives us

$$\mathbb{E}\left[\widetilde{T}\right] \leq T$$

and

$$\mathbb{E}\left[\widetilde{T}\right] \geq \sum_{\substack{uv: \\ l_u + l_v \leq \frac{2d^2L}{\varepsilon T}}} T_{uv}.$$

The right-hand side of the second expression is precisely the number of triangles τ in G such that $\rho(\tau) = uv$ with $l_u + l_u \leq \frac{2d^2L}{\varepsilon T}$. So let T^- be the number of triangles that do *not* satisfy this criterion. For each such triangle τ, there are at least $l_u + l_v$ updates in Σ to edges incident to $\rho(\tau)$. Furthermore, as the final graph has max degree d, at most $\binom{d}{2} \leq d^2/2$ triangles use any vertex. So we have:

$$\begin{aligned} L &\geq \frac{1}{2} \sum_v l_v \\ &\geq \frac{1}{d^2} \sum_{\substack{\tau, uv: \\ \rho(\tau) = uv}} l_u + l_v \\ &\geq \frac{1}{d^2} T^- \frac{2d^2L}{\varepsilon T} \end{aligned}$$

So $T^- \leq \varepsilon T/2$, and the result follows. □

Lemma 39.

$$Var(\widetilde{T}) \leq p^{-1}dT$$

Proof. For any fixed stream Σ, each $\widetilde{T}_e$ depends only on whether $h(e) = 1$, and so as h is pairwise independent, so are the $\widetilde{T}_e$, and so:

$$\begin{aligned}
\mathrm{Var}(\widetilde{T}) &= \sum_e \mathrm{Var}(\widetilde{T}_e) \\
&\leq \sum_e \mathbb{E}\left[\widetilde{T}_e^2\right] \\
&\leq \sum_e \mathbb{P}\left[h(e) = 1\right] p^{-2} T_e^2 \\
&\leq \sum_e p^{-1} d T_e \\
&= p^{-1} dT
\end{aligned}$$

□

Theorem 11. *There is a streaming algorithm for triangle counting in max-degree d graphs of length-L streams using $O\left(\frac{d^3L^2}{\varepsilon^2 T^2}\log n\right)$ bits of space.*

Proof. By Lemma 39, we may set p in the above algorithm to be $\frac{16d}{\varepsilon^2 T}$, so that the algorithm requires $O\left(\frac{d^3L^2}{\varepsilon^2T^2}\log n\right)$ space and $\mathrm{Var}(\widetilde{T}) = \frac{\varepsilon^2T^2}{16}$. Then, by Chebyshev's inequality, the probability that $|\widetilde{T} - \mathbb{E}\left[\widetilde{T}\right]| \geq \varepsilon T/2$ is at most $1/4$.

We may then repeat the algorithm $O(\log 1/\delta)$ times in parallel, taking the median, so that our final output is within $\varepsilon T/2$ of $\mathbb{E}\left[\widetilde{T}\right]$ with probability $1-\delta$. By Lemma 38, this implies it is within εT of T. □

7 Deterministic Turnstile-Sketching Equivalence

7.1 Overview

We will show that deterministic turnstile streaming algorithms can be expressed as linear sketches. Here these sketches will take the form of linear functions ϕ from $\mathbb{Z}^n$ to a module M whose elements can be stored in s space, where s is the space used by the turnstile streaming algorithm $\mathcal{A}$.

M and ϕ will be characterized by "moduli" a_i and "overflow vectors" o_i supported on indices smaller than i. A vector x in M is simply a vector in $\prod_{i=1}^n \mathbb{Z}_{a_i}$, but instead of addition being coordinatewise mod $(a_i)_n$, a coordinate i which becomes larger than a_i will "overflow", with o_i added to x for every time $a_i e_i$ has to be subtracted. This can cause repeated overflows, but as o_i is only supported on indices smaller than i, eventually these will stop.

By the structure theorem for $\mathbb{Z}$-modules, M is isomorphic to some direct product of cyclic modules, but this isomorphism is not (to our knowledge) necessarily calculable in small space. However, because our sketch ϕ represents a module, it has all the desirable properties of linear

sketches: it is mergeable, automatically allows deletions, and is indifferent to stream length and order.

We will start by defining M in terms of the parameters a_i and o_i, showing that if the parameters can be calculated in small space then the homomorphism can also be calculated in small space. We will then give two methods of generating these parameters, and show that the corresponding sketches can be used to solve stream problems, proving equivalence first for total functions:

Theorem 12. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of length $n+2s+2$, uses S space during updates and recovery, and uses s space between updates. If P corresponds to a* total function *on $\mathbb{Z}^n$, there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

Then, for algorithms that can tolerate very long stream lengths, we prove equivalence for general stream problems:

Theorem 2. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of all lengths, uses S space during updates and recovery, and uses s space between updates. Then there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

7.2 Our Module

7.2.1 Definition of M

Let $(a_i)_{i=1}^n$ be positive integers, and let at most m of them be greater than 1. Let $(o_i)_{i=1}^n$ be vectors such that for all i, $o_i \in \prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \{0\}^{n-i+1}$. We will define

$$M = \left(\prod_{i=1}^{n} \mathbb{Z}_{a_i}, \star \right)$$

a $\mathbb{Z}$-module with $\star$ as its addition operation. We will now recursively define a homomorphism $\phi : \mathbb{Z}^n \to M$, and then use this to define $\star$.

- $\phi(\mathbf{0}) = \mathbf{0}$
- For $i \in [n]$, and any vector $x + re_i$ where $x_j = 0$ for all $j \geq i$, $\phi(x + re_i) = (r \bmod a_i)e_i + \phi(x + (\lfloor r/a_i \rfloor)o_i)$.

This is well-defined because $x + (\lfloor r/a_i \rfloor)o_i$ is zero on all coordinates greater than $i - 1$.

We can now define $\star$ in terms of ϕ, using the fact that every vector in M is also a vector in $\mathbb{Z}^n$:

$$x \star y = \phi(x + y)$$

7.2.2 Algebraic Properties of M and ϕ

In this section we will prove that M is in fact a $\mathbb{Z}$-module, and ϕ is a homomorphism from $\mathbb{Z}^n$ to it.

Lemma 40. *ϕ is idempotent.*

Proof. As for any vector x in $\mathbb{Z}^n$ that is also in M, $\phi(x) = x$. □

Lemma 41. $\star$ *is commutative.*

Proof. By the symmetry of the definition. □

Lemma 42. $\star$ *is associative.*

Proof. We need to prove that for any x, y, z, $(x \star y) \star z = x \star (y \star z)$. As $x \star y = \phi(x + y)$ and we have already shown that $\star$ is commutative, it will suffice to prove that for all x, y, z, $\phi(\phi(x + y) + z) = \phi(x + y + z)$. We will prove this by induction on i, the smallest non-negative integer such that for all $j > i$, $x_j = y_j = z_j = 0$.

If $i = 0$, $x = y = z = \mathbf{0}$ and so as $\phi(\mathbf{0}) = \mathbf{0}$ the result follows immediately. Otherwise, suppose the result holds for $i - 1$ and let x, y, z be such that for all $j > i$, $x_j = y_j = z_j = 0$. Then we may write

$$\begin{aligned} x &= x' + r_1 e_i \\ y &= y' + r_2 e_i \\ z &= z' + r_3 e_i \end{aligned}$$

where x'_j, y'_j, z'_j are zero for all $j > i - 1$. Then, by the inductive hypothesis,

$$\begin{aligned} \phi(\phi(x + y) + z) &= \phi(\phi(x' + r_1 e_i + y' + r_2 e_i) + z' + r_3 e_i) \\ &= \phi(\phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i) + (r_1 + r_2 \bmod a_i) e_i + z' + r_3 e_i) \\ &= \phi(\phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i) + z' + \left\lfloor \frac{(r_1 + r_2 \bmod a_i) + r_3}{a_i} \right\rfloor o_i) \\ &+ (r_1 + r_2 + r_3 \bmod a_i) e_i \\ &= \phi(\phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i + z' + \left\lfloor \frac{(r_1 + r_2 \bmod a_i) + r_3}{a_i} \right\rfloor o_i)) \\ &+ (r_1 + r_2 + r_3 \bmod a_i) e_i \\ &= \phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i + z' + \left\lfloor \frac{(r_1 + r_2 \bmod a_i) + r_3}{a_i} \right\rfloor o_i) + (r_1 + r_2 + r_3 \bmod a_i) e_i \\ &= \phi(x' + y' + z' + \left\lfloor \frac{r_1 + r_2 + r_3}{a_i} \right\rfloor o_i) + (r_1 + r_2 + r_3 \bmod a_i) e_i \\ &= \phi(x' + y' + z' + (r_1 + r_2 + r_3) e_i) \\ &= \phi(x + y + z) \end{aligned}$$

as

$$x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i + z' + \left\lfloor \frac{(r_1 + r_2 \bmod a_i) + r_3}{a_i} \right\rfloor o_i$$

has zeros at every coordinate greater than $i - 1$. □

Lemma 43. $\forall x, y \in \mathbb{Z}^n, \phi(x + y) = \phi(x) \star \phi(y)$

Proof. We proceed by induction on i, the smallest non-negative integer such that $x_j = y_j = 0$ for all $j > i$. If $i = 0$, then $x = y = \mathbf{0}$ and so the result follows immediately. So suppose $i > 0$ and the result holds for all smaller i. Let $x = x' + r_1 e_i$, $y = y' + r_2 e_i$, where $x'_j = y'_j = 0$ for all $j \geq i$.

$$\begin{aligned}\phi(x+y) &= \phi(x' + y' + (r_1 + r_2)e_i) \\ &= \phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i) + (r_1 + r_2 \bmod a_i)e_i\end{aligned}$$

On the other hand:

$$\begin{aligned}\phi(x) \star \phi(y) &= \phi(\phi(x) + \phi(y)) \\ &= \phi(\phi(x' + \lfloor r_1/a_i \rfloor o_i) + \phi(y' + \lfloor r_2/a_i \rfloor o_i) + ((r_1 \bmod a_i) + (r_2 \bmod a_i))e_i) \\ &= \phi(\phi(x' + \lfloor r_1/a_i \rfloor o_i) + \phi(y' + \lfloor r_2/a_i \rfloor o_i) + \left\lfloor \frac{(r_1 \bmod a_i) + (r_2 \bmod a_i)}{a_i} \right\rfloor o_i) \\ &+ ((r_1 \bmod a_i) + (r_2 \bmod a_i) \bmod a_i)e_i \\ &= \phi(x' + \lfloor r_1/a_i \rfloor o_i) + \phi(y' + \lfloor r_2/a_i \rfloor o_i) + \phi(\left\lfloor \frac{(r_1 \bmod a_i) + (r_2 \bmod a_i)}{a_i} \right\rfloor o_i) \\ &+ (r_1 + r_2 \bmod a_i)e_i \\ &= \phi(x' + y' + (\lfloor r_1/a_i \rfloor + \lfloor r_2/a_i \rfloor + \left\lfloor \frac{(r_1 \bmod a_i) + (r_2 \bmod a_i)}{a_i} \right\rfloor)o_i) \\ &+ (r_1 + r_2 \bmod a_i)e_i \\ &= \phi(x' + y' + \left\lfloor \frac{r_1 + r_2}{a_i} \right\rfloor o_i) + (r_1 + r_2 \bmod a_i)e_i \\ &= \phi(x+y)\end{aligned}$$

□

Lemma 44. *$\star$ is invertible, with $\phi(-x)$ being the inverse of $\phi(x)$ for all $x \in \mathbb{Z}^n$.*

Proof. By the previous lemma,

$$\begin{aligned}\phi(x) \star \phi(-x) &= \phi(x + -x) \\ &= \phi(\mathbf{0}) \\ &= \mathbf{0}\end{aligned}$$

□

Therefore, M is an abelian group and so forms a $\mathbb{Z}$-module under the natural definition of integer multiplication.

Lemma 45. *$\phi : \mathbb{Z}^n \to M$ is a homomorphism of $\mathbb{Z}$-modules.*

Proof. We already have that ϕ preserves addition and multiplication by -1, so it must also preserve multiplication by elements of $\mathbb{Z}$. □

7.2.3 Space

In this section, we will prove that, provided the moduli a_i are small enough and can be generated along with the o_i in sufficiently small space, the sketch may be maintained in small space.

Theorem 46. *Suppose $\prod_{i=1}^{n} a_i \leq 2^s$, and for each i, a_i and o_i can be calculated in $O(S+s+m\log n)$ space. Then the sketch $\phi(x)$ can be stored in s space and maintained under updates to x using only $O(S+s+m\log n+\log r)$ space for updates to z of size r.*

We now present an algorithm for calculating $\phi(x)$. All vectors are stored as a list of indices and values.

Algorithm 3: Calculating $\phi(x)$

```
Calculate the moduli a_i.
z ← x
while ∃i, z_i ≥ a_i do
    Let i be the smallest index such that z_i ≥ a_i.
    Calculate o_i.
    z_i ← z_i − a_i
    z ← z + o_i
    Discard o_i.
end
return z
```

Lemma 47. *Algorithm 3 terminates.*

Proof. First note that, as o_i is only supported on indices smaller than i, for any j, z_j will not increase unless $i > j$, where i is the smallest index such that $z_i \geq a_i$ (or ∞ if there is no such index).

Now, we we prove that for any j from 0 to n, and any starting value of z, it will only take a finite number of iterations of the inner loop of the algorithm until the first time $i > j$. We will prove this by a double induction on j and z_j.

Suppose $j = 0$. Then $i > j$ at the start of the stream.

Suppose $j > 0$ and $z_j < a_j$, and the result holds for all smaller values of j. By the inductive hypothesis, after some finite number of iterations we reach the first time that $i > j - 1$. As this is the first time, z_j remains unchanged and so $i > j$.

Finally suppose $j > 0$, $z_j \geq a_j$, and the result holds for all j, z_j where at least one of j and z_j is smaller. By the inductive hypothesis, after some finite number of iterations we reach the first time that $i > j - 1$. At the next iteration, z_j is reduced by a_j and o_j is added to z. By applying the inductive hypothesis with this new value of z, a finite number more steps will bring us to the first time that $i < j$.

Therefore, by considering $j = n$ the algorithm will eventually terminate. □

Lemma 48. *When Algorithm 3 terminates, it returns $\phi(x)$.*

Proof. At the end of the algorithm, the output is $z = \phi(z)$, as $\forall i, z_i < a_i$. At the start of the algorithm $z = x$ and so $\phi(z) = \phi(x)$. So it will suffice to show that each iteration of the algorithm leaves $\phi(z)$ unchanged.

An iteration picks some i such that $z_i \geq a_i$ and replaces z with $z + o_i - a_i e_i$. So we need to show that $\phi(z + o_i - a_i e_i)$. By Lemma 45, ϕ is a homomorphism of $\mathbb{Z}$-modules. Therefore,

$$\begin{aligned}
\phi(z + o_i - a_i e_i) &= \phi(z) \star \phi(a_i e_i - o_i)^{-1} \\
&= \phi(z) \star ((a_i \bmod a_i) e_i + \phi(\lfloor a_i / a_i \rfloor o_i - o_i))^{-1} \\
&= \phi(z) \star (\mathbf{0})^{-1} \\
&= \phi(z)
\end{aligned}$$

concluding the proof. □

We now analyze the space complexity of updating this sketch. For the following lemmas, we will assume that the conditions of Theorem 46 hold. First we show that it is possible to store all the a_i simultaneously.

Lemma 49. *The moduli a_i can be stored in $O(s + m \log n)$ space.*

Proof. We can store the non-1 moduli as pairs (i, a_i). The indices take $O(\log n)$ bits to store, and the total space used by storing the values a_i is at most $\sum_{i=1}^{n} \log a_i = \log \prod_{i=1}^{n} a_i \leq s$. □

Lemma 50. *Algorithm 3 uses $O(S + s + m \log n + \sum_{i \in [n]: x_i > 0} \log x_i + ||x||_0 \log n)$ space.*

Proof. The space cost of the algorithm comes from calculating the moduli (which takes $O(S + s + m \log n)$ space), calculating o_i (which takes $O(S + s + m \log n)$ space), storing z, and performing addition on coordinates of z (with the things to be added of size at most that of a coordinate of o_i or a_i, and therefore always smaller than the size of some modulus a_j for $j \leq i$).

Therefore, it will suffice to show that storing z never requires more than $O(m \log n + \sum_{i \in [n]: x_i > 0} \log x_i + ||x||_0 \log n)$ space. First, note that a coordinate of z only increases when o_i is added to z, and this only happens when $z_j < a_j$ for every $j < i$. As each o_i is in $\prod_{j=1}^{n} \mathbb{Z}_{a_j}$, this has two implications:

1. At most $m + ||x||_0$ coordinates of z are ever non-zero.
2. Every non-zero coordinate z_j is either no larger than x_j, or is at most twice a_j.

The first of these two implies that we can store the indices j such that $z_j > 0$ in at most $O((m + ||x||_0) \log n)$ space, while the second implies that we can store the list of values associated with these indices in at most $O(\sum_{i=1}^{n} \log(2a_i) + \sum_{i \in [n]: x_i > 0} \log x_i) = O(s + \sum_{i \in [n]: x_i > 0} \log x_i)$ space.

□

Lemma 51. *For any x, y in M, $x \star y$ can be calculated in $O(S + s + m \log n)$ space.*

Proof. $x \star y = \phi(x + y)$, so as x and y are both in $\prod_{i=1}^{n} \mathbb{Z}_{a_i}$, this follows directly from the previous lemma. □

We are now ready to prove that, for suitably generated a_i and o_i, we may maintain our sketch in small space.

Theorem 46. *Suppose $\prod_{i=1}^{n} a_i \leq 2^s$, and for each i, a_i and o_i can be calculated in $O(S + s + m \log n)$ space. Then the sketch $\phi(x)$ can be stored in s space and maintained under updates to x using only $O(S + s + m \log n + \log r)$ space for updates to z of size r.*

Proof. We may store the sketch in only s space by only storing the indices i where $a_i > 1$. We can then query it in $O(S + s + m \log n)$ space by calculating the moduli, and update it in space $O(S + s + m \log n + \log r)$ for updates of size r to z by calculating $\phi(\phi(x) + re_i)$, where i is the coordinate updated. □

7.3 Sketching Total Functions

7.3.1 Overview

In order to prove an equivalence between linear sketches and turnstile algorithms for total functions, we need to define parameters a_i and o_i to instantiate the linear sketch $\phi \to \mathbb{Z}^n$.

Once we have defined these parameters we will prove the sketch is "correct" — for every $x \in \mathbb{Z}^n$, there is a stream with frequency x on which $\mathcal{A}$ outputs the same thing as it does on $\kappa(\phi(x))$. We will then show that it is possible to generate the parameters a_i and o_i in $O(s + m \log n)$ space, and therefore by Theorem 46 we may maintain the sketch in this space.

Finally, we will show that, using the streams described in the correctness section, it is possible to recover a solution to any stream problem solved by $\mathcal{A}$ using the sketch.

7.3.2 Defining the Parameters

The a_i and o_i will be defined as the output of the following procedure, which proceeds through the indices i with backtracking.

For $i = 1, \ldots, n$:

- Let x_j be defined as the j^{th} vector in $x \in \prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \mathbb{Z} \times \{0\}^{n-i}$ in little-endian order.

  Let j_2 be the smallest integer such that there exists $j_1 < j_2$ such that $\mathcal{A}(\kappa(x_{j_2})) = \mathcal{A}(\kappa(x_{j_1}))$. Choose a_i, o_i so that $x_{j_2} - x_{j_1} = a_i e_i - o_i$. Note that $a_i \geq 0$ as x_{j_2} is later than x_{j_1} in little-endian order. If $a_i > 0$, move on to the next i.

  If $a_i = 0$, let i' be the largest index such that $(x_{j_2} - x_{j_1})_{i'} > 0$. Choose $a_{i'}$ and $o_{i'}$ so that $x_{j_2} - x_{j_1} = a_{i'} e_{i'} - o_{i'}$, overwriting the old values of $a_{i'}$ and $o_{i'}$. Then roll i back to $i' + 1$ and continue from there.

Lemma 52. *This procedure will terminate after a finite number of steps.*

Proof. After each iteration, either i increases or i is set to $i' + 1$ with $a_{i'}$ reduced from its previous value. As the a_i take values in the positive integers, the second can only happen finitely many times, and so the procedure will eventually terminate. □

7.3.3 Space

Lemma 53. $\prod_{i=1}^{n} a_i \leq 2^s$.

Proof. Consider the procedure from Section 7.3.2. In the final iteration (that is, when a_n is defined rather than i rolling back to some earlier index), j_2 was the smallest integer such that there existed j_1 such that $\mathcal{A}(\kappa(x_{j_2})) = \mathcal{A}(\kappa(x_{j_1}))$, and $a_n = (x_{j_2} - x_{j_1})_n$.

As j_2 was the smallest integer such that this held, this implies that $\mathcal{A}(\kappa(x_0), \ldots, \mathcal{A}(\kappa(x_{j_2-1}))$ were all distinct states. As the sequence x_j comes from iterating through the vectors in $\prod_{i=1}^{n-1} \mathbb{Z}_{a_i} \times \mathbb{Z}$ in little-endian order, j_2 is at least $\prod_{i=1}^{n-1} a_i \times (x_{j_2})_n$. So as $a_n \leq (x_{j_2})_n$, there are at least $\prod_{i=1}^{n} a_i$ distinct states of $\mathcal{A}$, and so the result follows. □

Recall that $m = |\{i \in [n : a_i > 1\}|$.

Corollary 54. $m \leq s$

Proof. This follows from the fact that the procedure that generates the a_i will always roll back if it would generate an a_i equal to 0, and therefore all the a_i are positive integers. □

Lemma 55. *We may calculate all the moduli a_i in $O(s + m \log n)$ space.*

Proof. To execute the procedure that generates the a_i, we need to remember the values of all a_j for $j < i$ (which we can store in $O(s + m \log n)$ space, as at most m are greater than 1 and their magnitudes sum to at most 2^s), and we need to find the pair $j_2 > j_1$ such that $\mathcal{A}(\kappa(x_{j_2})) = \mathcal{A}(\kappa(x_{j_1}))$.

We can generate any $\kappa(x_j)$ we will need in $O(S + s)$ space given a list of the a_i, as they just require marching through the elements of $\prod_{i=1}^{j-1} \mathbb{Z}_{a_i} \times \mathbb{Z}$ in little-endian order while executing the state-transition function of $\mathcal{A}$, and the number of elements we go through is at most the number of distinct states of $\mathcal{A}$.

Therefore, we can find the pair in $O(S + s)$ space by running two copies of $\mathcal{A}$ and feeding them the streams $\kappa(x_j)$ until we find a collision. □

Lemma 56. *For any i, o_i can be calculated in $O(S + s + m \log n)$ space.*

Proof. First note that, as each o_i is in $\prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \mathbb{Z}^{n-i+1}$, they can be *stored* in $O(s + m \log n)$ space by storing $(j, (o_i)_j)$ pairs as above.

To calculate o_i, we may first calculate all the a_i as above, and then run the procedure until the final time where it changes a_i. At that point we may read off o_i (as we know x_j and $x_{j'}$). □

7.3.4 Correctness

Theorem 57. *Let $f : \mathbb{Z}^n \to \{0, 1\}$ be any function. Suppose there is a "post-processing" function g such that, for all σ of length at most $n + 2m + 2$, $g(\mathcal{A}(\sigma)) = f(\operatorname{freq} \sigma)$. Then for all $x \in \mathbb{Z}^n$, $f(x) = g(\mathcal{A}(\kappa(\phi(x))))$.*

Proof. We proceed by induction on i, the largest non-negative integer such that $x_j < a_j$ for all $j > i$, and x_i.

Suppose $i = 0$. Then $x = \phi(x)$ and the result follows immediately, as $\kappa(x)$ has length at most n. So suppose that this is not the case, and the result holds for all x with smaller i or the same i and smaller x_i.

Then by the construction of o_i above, there exist x and y in $\prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \{0\}^{n-i+1}$ and integers $r' < r$ such that $\mathcal{A}(\kappa(x + re_i)) = \mathcal{A}(\kappa(y + r'e_i))$, and $o_i = y - x$, $a_i = r - r'$.

Now write $x = x' + x_i e_i + x''$, where x' is zero on all indices at least i and x'' is zero on all indices no greater than i. Then

$$\begin{aligned}\phi(x) &= \phi(x' + \lfloor x_i/a_i \rfloor o_i) + (x_i \bmod a_i)e_i + x'' \\ &= \phi(x' + o_i + (x_i - a_i)e_i + x'')\end{aligned}$$

and so by the inductive hypothesis:

$$g(\mathcal{A}(\kappa(\phi(x)))) = f(x' + o_i + (x_i - a_i)e_i + x'')$$

Now consider the following two streams:

$$\begin{aligned}\sigma_1 &= \kappa(x + re_i) \cdot \kappa(-x - re_i) \cdot \kappa(x) \\ \sigma_2 &= \kappa(y + r'e_i) \cdot \kappa(-x - re_i) \cdot \kappa(x)\end{aligned}$$

Note that x and y are both supported on at most m indices, so the length of these streams is at most $n + 2m + 2$ and so $g(\mathcal{A}(\sigma_1)) = f(\text{freq}\,\sigma_1)$ and $g(\mathcal{A}(\sigma_2)) = f(\text{freq}(\sigma_2))$. Furthermore, as $\mathcal{A}(\kappa(x + re_i)) = \mathcal{A}(\kappa(y + r'e_i))$, $\mathcal{A}(\sigma_1) = \mathcal{A}(\sigma_2)$, and so $f(\text{freq}(\sigma_1) = f(\text{freq}(\sigma_2))$.

Now $\text{freq}(\sigma_1) = x$, while

$$\begin{aligned}\text{freq}(\sigma_2) &= (y - x) - (r - r')e_i + x \\ &= o_i - a_i e_i + x \\ &= x' + o_i + (x_i - a_i)e_i + x''\end{aligned}$$

and so $f(x' + o_i + (x_i - a_i)e_i + x'') = f(x)$, and so

$$g(\mathcal{A}(\kappa(\phi(x)))) = f(x),$$

completing the proof. □

7.3.5 Turnstile-Sketching Equivalence

Theorem 12. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of length $n+2s+2$, uses S space during updates and recovery, and uses s space between updates. If P corresponds to a* total function *on $\mathbb{Z}^n$, there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

Proof. Let $\mathcal{A}$ be the original algorithm. The algorithm will be to keep $\phi(x)$, where x is the input vector (which by the previous sections we can do in $O(s + m \log n) \leq O(s \log n)$ space), and then give $\mathcal{A}$ $\kappa(\phi(x))$. By Theorem 57, as $m \leq s$, the output of $\mathcal{A}$ will be $f(x)$.

By the lemmas in Section 7.3.3, the conditions of Theorem 46 hold, and so this sketch can be stored in s space, and maintained in $O(S + s + m \log n) \leq O(S + s \log n)$ space (as $m \leq s$ by Corollary 54). Recovering $f(x)$ from the sketch requires running $\mathcal{A}$ on $\kappa(x)$, which takes $O(S + s)$ space. □

7.4 Sketching General Stream Problems

7.4.1 Overview

In order to prove an equivalence between linear sketches and turnstile algorithms for general stream problems, we need to define parameters a_i and o_i to instantiate the linear sketch $\phi \to \mathbb{Z}^n$.

Once we have defined these parameters we will prove the sketch is "correct" — for every $x \in \mathbb{Z}^n$, there are streams with frequency $x, \phi(x)$ on which $\mathcal{A}$ outputs the same thing. We will then show that it is possible to generate the parameters a_i and o_i in $O(s + m \log n)$ space, and therefore by Theorem 46 we may maintain the sketch in this space.

Finally, we will show that, using the streams described in the correctness section, it is possible to recover a solution to any stream problem solved by $\mathcal{A}$ using the sketch.

7.4.2 Defining the Parameters

Along with the parameters a_i and o_i, we also define "prefix vectors" π_i for $i = 0, \ldots, n$ and "covering streams" τ_i^x (for $x \in \prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \mathbb{Z} \times \{0\}^{n-i}$ and $i = 1, \ldots, n$) to be used in the recursive construction and in the later proof of correctness.

These will be defined as the output of the following procedure, which proceeds through the indices i with backtracking.

Let π_0 be the empty stream. For $i = 1, \ldots, n$:

- We start by defining the covering streams τ_i^x. Let x_j be defined as the j^{th} vector in $x \in \prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \mathbb{Z} \times \{0\}^{n-i}$ in little-endian order. Then we define $\tau_i^{x_1} = \tau_i^0 = \pi_{i-1}$. For $j > 0$, we define $\tau_i^{x_j} = \tau_i^{x_{j-1}} \cdot \kappa(x_j - x_{j-1})$.

  Note that for any $j_1 < j_2$, as $\tau_i^{x_{j_1}}$ is a prefix of $\tau_i^{x_{j_2}}$ we may write $\tau_i^{x_{j_2}} = \tau_i^{x_{j_1}} \cdot \alpha$ for some stream α and $\operatorname{freq} \alpha$ will be equal to $x_{j_2} - x_{j_1}$.

- Let j be the smallest integer such that there exists $j' < j$ such that $\mathcal{A}(\tau_i^{x_j}) = \mathcal{A}(\tau_i^{x_{j'}})$. Choose a_i, o_i so that $x_j - x_{j'} = a_i e_i - o_i$. Note that $a_i \geq 0$ as x_j is later than $x_{j'}$ in little-endian order. If $a_i > 0$, set $\pi_i = \tau_i^{x_{j'}}$ and move on to the next i.

  If $a_i = 0$, let i' be the largest index such that $(x_j - x_{j'})_{i'} > 0$. Choose $a_{i'}$ and $o_{i'}$ so that $x_j - x_{j'} = a_{i'} e_{i'} - o_{i'}$, and set $\pi_{i'} = \tau_i^{x_{j'}}$, overwriting the old values of $a_{i'}$, $o_{i'}$, and $\pi_{i'}$. Then roll i back to $i' + 1$ and continue from there.

Lemma 58. *This procedure will terminate after a finite number of steps.*

Proof. After each iteration, either i increases or i is set to $i' + 1$ with $a_{i'}$ reduced from its previous value. As the a_i take values in the positive integers, the second can only happen finitely many times, and so the procedure will eventually terminate. □

7.4.3 Space

Lemma 59. $\prod_{i=1}^{n} a_i \leq 2^s$.

Proof. Consider the procedure from Section 7.4.2. In the final iteration (that is, when a_n is defined rather than i rolling back to some earlier index), j was the smallest integer such that there existed j' such that $\mathcal{A}(\tau_n^{x_j}) = \mathcal{A}(\tau_n^{x'_j})$, and $a_n = (x_j - x_{j'})_n$.

As j was the smallest integer such that this held, this implies that $\mathcal{A}(\tau_n^{x_0}), \ldots, \mathcal{A}(\tau_n^{x_{j-1}})$ were all distinct states. As the sequence x_k comes from iterating through the vectors in $\prod_{i=1}^{n-1} \mathbb{Z}_{a_i} \times \mathbb{Z}$ in little-endian order, j is at least $\prod_{i=1}^{n-1} a_i \times (x_j)_n$. So as $a_n \leq (x_j)_n$, there are at least $\prod_{i=1}^{n} a_i$ distinct states of $\mathcal{A}$, and so the result follows. □

Recall that $m = |\{i \in [n : a_i > 1\}|$.

Corollary 60. $m \leq s$

Proof. This follows from the fact that the procedure that generates the a_i will always roll back if it would generate an a_i equal to 0, and therefore all the a_i are positive integers. □

Lemma 61. *We may calculate all the moduli a_i, while generating the stream π_n, in $O(S + s + m \log n)$ space.*

Proof. To execute the procedure that generates the a_i, we need to remember the values of all a_j for $j < i$ (which we can store in $O(s + m \log n)$ space, as at most m are greater than 1 and their magnitudes sum to at most 2^s), we need to remember $\mathcal{A}(\pi_{i-1})$ (which takes $O(s)$ space) and then we need to find the pair $j > j'$ such that $\mathcal{A}(\tau_i^{x_j}) = \mathcal{A}(\tau_i^{x_{j'}})$.

Given the moduli $(a_j)_{j=1}^{i}$, we can generate the elements of the streams $\tau_i^{x_j}$ (from after π_{i-1}) on the fly in $O(S + s)$ space, as they just require marching through the elements of $\prod_{j=1}^{i-1} \mathbb{Z}_{a_i} \times \mathbb{Z}$ in little-endian order while executing the transition function of $\mathcal{A}$ on each update, and the number of elements we go through is at most the number of distinct states of $\mathcal{A}$.

Therefore, we can find the pair in $O(S + s)$ space by running two copies of $\mathcal{A}$ and feeding them the streams $\tau_i^{x_j}$ until we find a collision. □

Lemma 62. *For any i, o_i can be calculated in $O(S + s + m \log n)$ space.*

Proof. First note that, as each o_i is in $\prod_{j=1}^{i-1} \mathbb{Z}_{a_j} \times \mathbb{Z}^{n-i+1}$, they can be *stored* in $O(s + m \log n)$ space by storing $(j, (o_i)_j)$ pairs as above.

To calculate o_i, we may first calculate all the a_i as above, and then run the procedure until the final time where it changes a_i. At that point we may read off o_i (as we know x_j and $x_{j'}$, as we tracked them while generating the streams $\tau_i^{x_j}$ and $\tau_i^{x_{j'}}$). □

7.4.4 Correctness

Lemma 63. *Let α, β be any pair of streams. Then there are infinitely many $l \in \mathbb{N}$ such that $\mathcal{A}(\alpha \cdot \beta^l) = \mathcal{A}(\alpha \cdot \beta^{2^s})$.*

Proof. Consider the sequence of states $q_l = \mathcal{A}(\alpha \cdot \beta^l)$. As there are only 2^s distinct states, there is some state that recurs infinitely many times, and that state must appear for some $l \leq 2^s$. So let this $l = 2^s - k$. Each time this state appears, $\mathcal{A}(\alpha \cdot \beta^{2^s})$ appears k states later. So $\mathcal{A}(\alpha \cdot \beta^{2^s})$ also appears infinitely many times. □

Theorem 64. *For all $x \in \mathbb{Z}^n$, there is a stream σ such that* $\operatorname{freq}\sigma = x$ *and:*

$$\mathcal{A}(\sigma) = \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_n - a_ne_n)^{2^s} \cdot \kappa(\phi(x)))$$

Proof. For each $i \in [n]$, let ψ_i be such that $\pi_i = \pi_{i-1} \cdot \psi_i$ (recall that each π_i is a prefix of the next), and let ρ_i be the stream found in the construction of M such that $\mathcal{A}(\pi_i \cdot \rho_i) = \mathcal{A}(\pi_i)$ and $\operatorname{freq}\rho_i = a_ie_i - o_i$. For $y \in \mathbb{N}^n$, let ξ_y be the following stream:

$$\psi_1 \cdot \rho_i^{y_1} \dots \psi_n \cdot \rho_n^{y_n}$$

Then for all $y \in \mathbb{N}^n$,

$$\mathcal{A}(\xi_y) = \mathcal{A}(\psi_1 \dots \psi_n) = \mathcal{A}(\pi_n)$$

while $\operatorname{freq}\xi_y = \operatorname{freq}\pi_n + \sum_{i=1}^n y_i(a_ie_i - o_i)$. Next, for $y \in \mathbb{N}^n$, let

$$\chi_y = \kappa(o_1 - a_1e_1)^{y_1} \dots \kappa(o_n - a_ne_n)^{y_n}$$

so $\operatorname{freq}\chi_y = -\sum_{i=1}^n y_i(a_ie_i - o_i)$. We will prove the theorem for a σ of the form

$$\sigma = \xi_y \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x))$$

for carefully chosen y and z. Note that

$$\begin{aligned}\operatorname{freq}\sigma &= \operatorname{freq}\xi_y - \operatorname{freq}\pi_n + \operatorname{freq}\chi_z + \phi(x) \\ &= \phi(x) + \sum_{i=1}^n (y_i - z_i)(a_ie_i - o_i). \end{aligned} \tag{1}$$

In particular, we will choose z such that

$$\mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{z_1} \dots \kappa(o_i - a_ie_i)^{z_i}) = \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_i - a_ie_i)^{2^s})$$

for each $i \in [n]$.

We show that the theorem holds for such a σ and z by induction on i, the largest non-negative integer such that $0 \le x_j < a_j$ for all $j > i$.

Suppose $i = 0$. Then $x = \phi(x)$, so we can take $\sigma = \xi_y \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x))$, where both y and z are the vectors with 2^s in every coordinate. Then

$$\begin{aligned}\mathcal{A}(\sigma) &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x)) \\ &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_n - a_ne_n)^{2^s} \cdot \phi(x))\end{aligned}$$

and by (1), $\operatorname{freq}\sigma = \phi(x) = x$. Finally, the condition on z is trivially satisfied, as $z_i = 2^s$ for each i.

Now suppose $i > 0$, and the result holds for all x with smaller i. Write $x = x' + x_ie_i + x''$, where x' is zero on all indices at least i and x'' is zero on all indices no greater than i. Then

$$\begin{aligned}\phi(x) &= \phi(x' + \lfloor x_i/a_i \rfloor o_i) + (x_i \bmod a_i)e_i + x'' \\ &= \phi(x' + \lfloor x_i/a_i \rfloor o_i + (x_i \bmod a_i)e_i + x'')\end{aligned}$$

and by the inductive hypothesis there exists a $\sigma' = \xi_{y'} \cdot \overline{\pi_n} \cdot \chi_{z'} \cdot \kappa(\phi(x))$ such that

$$\operatorname{freq}\sigma' = x' + \lfloor x_i/a_i \rfloor o_i + (x_i \bmod a_i)e_i + x''$$

and

$$\mathcal{A}(\sigma') = \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_n - a_ne_n)^{2^s} \cdot \kappa(\phi(x)))$$

with z' such that

$$\mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{z'_1} \dots \kappa(o_j - a_je_j)^{z'_j}) = \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_j - a_je_j)^{2^s})$$

for each $j \in [n]$.

Now, by Lemma 63, there are infinitely many $l \in \mathbb{N}$ such that

$$\begin{aligned}&\mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_{i-1} - a_{i-1}e_{i-1})^{2^s} \cdot \kappa(o_i - a_ie_i)^l)\\ =&\mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_{i-1} - a_{i-1}e_{i-1})^{2^s} \cdot \kappa(o_i - a_ie_i)^{2^s})\end{aligned}$$

so let l be such that this holds and $l \geq z'_i - \lfloor x_i/a_i \rfloor$. We will define z to be z' at every coordinate except that $z_i = l$. We will define y to be y' except with $y_i = y'_i + l + \lfloor x_i/a_i \rfloor - z'_i$, so y is still in $\mathbb{N}^n$.

Now let $\sigma = \xi_y \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x))$. We will show this satisfies all the conditions required by the inductive hypothesis. First, we show that z obeys the desired property. For all $j \in [n]$, if $j < i$ it holds by the inductive hypothesis, as $z_j = z'_j$ for all $j < i$. Then, if $j = i$,

$$\begin{aligned}\mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{z_1} \dots \kappa(o_j - a_je_j)^{z_j}) &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_{i-1} - a_{i-1}e_{i-1})^{2^s}) \cdot \kappa(o_i - a_ie_i)^l)\\ &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1e_1)^{2^s} \dots \kappa(o_i - a_ie_i)^{2^s})\end{aligned}$$

using the $j < i$ property and our choice of l. For $j > i$, the result again holds by the inductive hypothesis, as it holds for $j = i$ and $z_j = z'_j$ for all $j > i$.

Now we show that $\mathcal{A}(\sigma)$ takes the correct value.

$$\begin{aligned}\mathcal{A}(\sigma) &= \mathcal{A}(\xi_y \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x)))\\ &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \chi_z \cdot \kappa(\phi(x)))\\ &= \mathcal{A}(\pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1)^{2^s} \dots \kappa(o_n - a_n)^{2^s} \cdot \kappa(\phi(x)))\end{aligned}$$

by the property we just proved for z.

Finally we need to prove that $\operatorname{freq}\sigma = x$. The difference between σ and σ' is that we replaced $\xi_{y'}$ with ξ_y and $\chi_{z'}$ with χ_z, and y, z each differ from y', z' only in coordinate i. Therefore by (1),

$$\begin{aligned}\operatorname{freq}\sigma &= \operatorname{freq}\sigma' + (y_i - y'_i + z'_i - z_i)(a_ie_i - o_i)\\ &= (x' + \lfloor x_i/a_i \rfloor o_i + (x_i \bmod a_i)e_i + x'') + (l + \lfloor x_i/a_i \rfloor - z_i + z'_i - l)(a_ie_i - o_i)\\ &= x' + (\lfloor x_i/a_i \rfloor a_i + (x_i \bmod a_i))e_i + x''\\ &= x' + x_ie_i + x''\\ &= x\end{aligned}$$

completing the proof. □

7.4.5 Sketching-Turnstile Equivalence

Theorem 2. *Suppose there is a deterministic algorithm solving a streaming problem P that works on streams of all lengths, uses S space during updates and recovery, and uses s space between updates. Then there is a linear sketching algorithm for P that uses $O(S + s \log n)$ space during updates and recovery, and stores an s space sketch.*

Proof. Let $\mathcal{A}$ be the original algorithm. The new algorithm will be to construct M and ϕ as above, and as we receive updates to the input vector x, maintain $\phi(x)$. By the Lemmas in Section 7.4.3, the conditions of Theorem 46 are satisfied, so this will require $O(S + s + m \log n) \leq O(S + s \log n)$ space to compute (as $m \leq s$ by Corollary 60).

Then, at the end of the stream, we will input $\sigma^* := \pi_n \cdot \overline{\pi_n} \cdot \kappa(o_1 - a_1 e_1)^{2^s} \dots \kappa(o_n - a_n e_n)^{2^s} \cdot \kappa(\phi(x))$ to $\mathcal{A}$, and output whatever $\mathcal{A}$ recovers from the resulting state (as we can compute π_n we can also compute $\overline{\pi_n}$). By Theorem 64, there is a stream σ with $\operatorname{freq} \sigma = x$ such that $\mathcal{A}(\sigma^*) = \mathcal{A}(\sigma)$, so as $\mathcal{A}$ would have output a correct answer for σ it will output the same correct answer when given σ^*.

This recovery algorithm takes $O(S + s + m \log n) \leq O(S + s \log n)$ space, as by Lemma 61 we can generate π_n in that space (and therefore $\overline{\pi_n}$), even though we could not *store* the whole stream. Similarly, we can generate the streams $\kappa(o_i - a_i)^{2^s}$ by generating a_i and o_i and using an s-bit counter to insert it the correct number of times. Other than computing the stream, we simply maintain $\mathcal{A}$ under the stream σ^* and apply the recovery algorithm, both of which use S space by assumption. □

References

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. *SODA*, pages 459–467, 2012.

[AHLW16] Yuqing Ai, Wei Hu, Yi Li, and David P Woodruff. New characterizations in turnstile streams with applications. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742. SIAM, 2017.

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1345–1364. SIAM, 2016.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

[CCF02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[CDIM03] Graham Cormode, Mayur Datar, Piotr Indyk, and S Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, 15(3):529–540, 2003.

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[FIS08] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

[FS05] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 209–217. ACM, 2005.

[Gan08] Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *International Computer Science Symposium in Russia*, pages 204–215. Springer, 2008.

[HLY19] Kaave Hosseini, Shachar Lovett, and Grigory Yaroslavtsev. Optimality of linear sketching under modular updates. *CCC*, 2019.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[IP11] Piotr Indyk and Eric Price. K-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 627–636. ACM, 2011.

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716. Springer, 2005.

[JW18] Rajesh Jayaram and David P Woodruff. Data streams with bounded deletions. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 341–354. ACM, 2018.

[KKM13] Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1131–1142. Society for Industrial and Applied Mathematics, 2013.

[KKP18] John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 556–567. IEEE, 2018.

[KLM+14] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *FOCS*, 2014.

[KMY18] Sampath Kannan, Elchanan Mossel, and Grigory Yaroslavtsev. Linear sketching over $\mathbb{F}_2$. *CCC*, 2018.

[Kon15] Christian Konrad. Maximum matching in turnstile streams. In *Algorithms-ESA 2015*, pages 840–852. Springer, 2015.

[KP17] John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1778–1797. SIAM, 2017.

[LNW14] Yi Li, Huy L. Nguyễn, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 174–183, 2014.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NY19] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1844–1860. SIAM, 2019.

[PT12] Rasmus Pagh and Charalampos E Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

[PTTW13] A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, September 2013.

[PW12] Eric Price and David P Woodruff. Applications of the shannon-hartley theorem to data streams and sparse recovery. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 2446–2450. IEEE, 2012.

[TKMF09] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846. ACM, 2009.