Revisiting Frequency Moment Estimation in Random Order Streams

Vladimir Braverman[1]
Johns Hopkins University
vova@cs.jhu.edu

Emanuele Viola
Northeastern University
viola@ccs.neu.edu

David P. Woodruff
Carnegie Mellon University
dwoodruf@cs.cmu.edu

Lin F. Yang[2]
Princeton University
lin.yang@princeton.edu

Abstract

We revisit one of the classic problems in the data stream literature, namely, that of estimating the frequency moments F_p for $0 < p < 2$ of an underlying n-dimensional vector presented as a sequence of additive updates in a stream. It is well-known that using p-stable distributions one can approximate any of these moments up to a multiplicative $(1+\epsilon)$-factor using $O(\epsilon^{-2}\log n)$ bits of space, and this space bound is optimal up to a constant factor in the turnstile streaming model. We show that surprisingly, if one instead considers the popular random-order model of insertion-only streams, in which the updates to the underlying vector arrive in a random order, then one can beat this space bound and achieve $\tilde{O}(\epsilon^{-2}+\log n)$ bits of space, where the $\tilde{O}$ hides $\mathrm{poly}(\log(1/\epsilon)+\log\log n)$ factors. If $\epsilon^{-2} \approx \log n$, this represents a roughly quadratic improvement in the space achievable in turnstile streams. Our algorithm is in fact deterministic, and we show our space bound is optimal up to $\mathrm{poly}(\log(1/\epsilon)+\log\log n)$ factors for deterministic algorithms in the random order model. We also obtain a similar improvement in space for $p = 2$ whenever $F_2 \gtrsim \log n \cdot F_1$.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Data Stream, Frequency Moments, Random Order, Space Complexity, Insertion Only Stream

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.25

Related Version A full version of the paper can be found at `https://arxiv.org/abs/1803.02270`.

[1] This material is based upon work supported in part by the National Science Foundation under Grants No. 1447639, 1650041 and 1652257, Cisco faculty award, and by the ONR Award N00014-18-1-2364.
[2] This material is based upon work supported in part by National Science Foundation under Grant No. 1447639, 1650041 and 1652257. Work was done while the author was at Johns Hopkins University.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Analyzing massive datasets has become an increasingly challenging problem. Data sets, such as sensor networks, stock data, web/network traffic, and database transactions, are collected at a tremendous pace. Traditional algorithms that store an entire dataset in memory are impractical. The *streaming model* has emerged as an important model for coping with massive datasets. Streaming algorithms are typically randomized and approximate, making a single pass over the data and using only a small amount of memory.

A well-studied problem in the streaming model is that of estimating the frequency moments of an underlying vector. Formally, in an *insertion-only stream*, the algorithm is presented with a sequence of integers $\langle a_1, a_2, \ldots, a_m \rangle$ from a universe $[n]$. A *turnstile stream* is defined similarly except integers may also be removed from the stream. The p-th frequency moment of the stream is defined to be $F_p = \sum_{i \in [n]} f_i^p$, where f_i is number of times integer i occurs in the stream, i.e., its frequency. The quantity F_p is a basic, yet very important statistic of a dataset (e.g. [1]). For example, interpreting 0^0 as 0, F_0 is equal to the number of distinct items in the stream. F_2 measures the variance and can be used to estimate the size of a self-join in database applications. It also coincides with the (squared) Euclidean norm of a vector and has applications in geometric and linear algebraic problems on streams. For other non-integer $p > 0$, F_p can serve as a measure of the entropy or skewness of a dataset, which can be useful for query optimization.

In their seminal paper, Alon, Matias & Szegedy [2] introduced the study of frequency moments in the streaming model. Nearly two decades of research have been devoted to the space and time complexity of this problem. An incomplete list of papers on frequency moments includes [3, 8, 11, 18–20, 23], and [7]; please also see the references therein. In the turnstile model, $\Theta(\epsilon^{-2} \log(mn))$ bits of space is necessary and sufficient for a randomized one-pass streaming algorithm to obtain a $(1 \pm \epsilon)$-approximation to F_p for $0 < p \leq 2$ [21]. Here, by $(1 \pm \epsilon)$-approximation, we mean that the algorithm outputs a number $\tilde{F}_p$ for which $(1 - \epsilon)F_p \leq \tilde{F}_p \leq (1 + \epsilon)F_p$. For larger values of p, i.e., $p > 2$, the memory required becomes polynomial in n rather than logarithmic [5, 11, 19].

In this paper, we study the frequency moment estimation problem in the *random-order model*, which is a special case of insertion streams in which the elements in the stream occur in a uniformly random order and the algorithm sees the items in one pass over this random order. This model was initially studied by Munro and Paterson [22], which was one of the initial papers on data streams in the theory community. Random-order streams occur in many real-world applications and are studied in, e.g., [10, 13, 15] and the references therein. It has been shown that there is a considerable difference between random order streams and general arbitrary order insertion streams for problems such as quantile estimation [10, 16]. Notice that [12] studies the frequency moment problem for stochastic streams, in which data points are generated from some distribution. That model is different from ours, but when conditioning on the realizations of the values of each point, the stream is exactly in random order. Therefore, our upper bounds are applicable to that model as well.

However, there is a gap in our understanding for the important problem of F_p-estimation, $0 < p \leq 2$, in the random order model. On the one hand there is an $\Omega(\epsilon^{-2})$ bits of space lower bound [9]. On the other hand, the best upper bound we have is the same as the best upper bound in the turnstile model, namely $O(\epsilon^{-2} \log n)$ bits [2, 20, 21]. In practice it would be desirable to obtain $O(\epsilon^{-2} + \log n)$ bits rather than $O(\epsilon^{-2} \log n)$ bits, since if ϵ is very small this can lead to considerable savings. For example, if $\epsilon^{-2} \approx \log n$, it would represent a roughly quadratic improvement. The goal of this work is to close this gap in our understanding of the memory required for F_p-estimation, $0 < p \leq 2$, in the random order model. Note that in all our bounds, we apply the convention that $m = \text{poly}(n)$.

1.1 Our Contribution

In this paper, we make considerable progress on understanding the space complexity of F_p estimation with $0 < p \leq 2$, in random order streams. Specifically,

- for F_2, we show that there exists a simple, and in fact deterministic, one-pass algorithm using $\mathcal{O}(\epsilon^{-2}+\log n)$ bits of space to output a $(1\pm\epsilon)$ approximation, provided $F_2 \geq m\cdot\log n$,
- for F_p with $p \in (0,2)\backslash\{1\}$, we obtain a one-pass deterministic algorithm to obtain a $(1 \pm \epsilon)$ approximation using $\tilde{O}(\epsilon^{-2} + \log n)$ bits of space[3]. We also show that this space complexity is optimal for deterministic algorithms in the random order model, up to $\text{poly}(\log(1/\epsilon)) + \text{poly}(\log\log n)$ factors. Note that for the case $p = 1$, F_p is the length of the stream, which can be computed using $O(\log n)$ bits of memory.

1.2 Our Techniques

For F_2, we partition the stream updates into a sequence of small blocks for which each block can be stored using $\mathcal{O}(\epsilon^{-2})$ bits. We then construct an unbiased estimator by counting the number of "pairs" of updates that belong to the same universe item. The counting can be done exactly and with only $\mathcal{O}(\log n)$ bits of space in each small block. We further show that if $F_2 \geq \log n \cdot F_1$, we can obtain the desired concentration by averaging the counts over all blocks. The analysis of the concentration is by constructing a Doob martingale over the pairs and applying Bernstein's inequality.

For F_p with $0 < p < 2$ our algorithm is considerably more involved. We first develop a new reduction from estimating F_p to finding ℓ_p heavy hitters (an ℓ_p heavy hitter is an item with frequency comparable to $F_p^{1/p}$ of the stream) and then we establish the heavy hitter algorithm. Both our new reduction and the heavy hitter-finding algorithm are redesigned over the existing ones to allow us to obtain better space complexity. Our heavy hitter finding algorithm is similar to that of [6]. However, we need to be more careful so that the final space bound of the heavy hitter algorithm can be controlled (e.g., we cannot afford to store $\Omega(1/\epsilon^2)$ many coordinate IDs, which would cost $\Omega((\log n)/\epsilon^2)$ bits of space. We have to store only $O(1/\epsilon^2)$ many approximate values). The major contribution of our algorithm is the careful reduction from F_p estimation to ℓ_p heavy hitter-finding. Many reductions are known in the literature but are suffering from a $\text{poly}(\log n)$ space blow up. Our careful reduction allows us to pay only a $\text{poly}(\log\log n)$ space blow up.

In what follows, we illustrate the high level ideas of the heavy hitter reduction. Let $v = (f_1, f_2, \ldots, f_n)$ be the frequency vector of the items. We first apply a random scaling X_i to each f_i, where each X_i is pairwise independently drawn from some distribution. We argue that finding the leading ℓ_p heavy hitter of the scaled frequency vector (denoted as $X_{i^*}f_{i^*}$) gives a good estimation of the $F_p^{1/p}$ of the original stream. The distribution of X_i is a so-called p-inverse distribution (see Definition 2 for details). This distribution has a similar tail as that of the max-stable distributions used in [3] or the precision sampling distribution in [4]. However, it has different properties better suited to our setting, e.g., we can use pairwise independent variables to do the scaling. In contrast, for max-stable distributions, we have to use fully random hash functions and apply pseudo-random generators, which is problematic in our very low space regime (a related technique called tabulation-based hashing has similar problems). Note that [4] does not require pseudo-random generators, but their precision sampling technique aims to solve a more general family of problems and their

[3] We use $\tilde{O}$ to hide $\text{poly}(\log\log n + \log\epsilon^{-1})$ factors.

distribution has a slightly different form, e.g., the random variable is drawn from a continuous distribution. The p-inverse distribution is particularly suited for p-norm estimation, and allows for a concise algorithm and improved space complexity in our setting.

After choosing the random scalings, we group the coordinates of v into $\Theta(\log n)$ levels by their scalings, i.e., if $2^{-w}F_p < X_i^p \leq 2^{-w+1}F_p$, then i is in level w for some integer w. Since we do not know F_p before the stream arrives, the exact level ID is not known to the algorithm. But since each X_i is obtained pseudo-randomly, the algorithm is able to know whether two coordinates are in a same level or not (i.e., the relative level ID) before looking at the stream. Let $Z_w \subset [n]$ be the set of universe items in level w. We observe that if for some coordinate $i^* \in Z_w$, it satisfies $X_{i*}^p f_{i^*}^p \approx F_p$, then $f_{i^*}^p = \Omega(2^w)$. Fortunately, we can also show that in expectation, i^* is an ℓ_p-heavy hitter of the substream induced by Z_w (i.e., in expectation the F_p of the stream restricted to Z_w is approximately 2^w). Our algorithm simply *looks for* i^* *from* Z_w *for every* $w \in [\log n]$. One may notice that if we run the search instance in parallel, then there will be a $(\log n)$-factor blowup in the space of a heavy hitter algorithm. However, we can show that for random order streams, one can choose a $w_0 = \Theta(\log \log n)$ such that

1. for all $w > w_0$: the search for the heavy hitter i^* can be done in one pass and in sequence for each w. This is because f_{i^*} is large (i.e., $\Omega(2^w)$) and a small portion of the random order stream contains sufficient information about i^*.
2. 2. for all $w \leq w_0$: with high probability, $|Z_w| = \text{poly}(\log n)$. We thus do a brute force search for each level w below w_0 in parallel. Each search instance uses a small amount of memory because of the small universe size.

The final space overhead becomes a $\text{poly}(w_0)$ factor rather than a $\Theta(\log n)$ factor. This observation is critical to reduce space usage for approximating frequency moments in random order streams and is, to the best of our knowledge, new. For $p \geq 2$, the above claim is no longer true. We leave the exact space complexity for $p \geq 2$ as an open problem.

1.3 Roadmap

In Section 2, we introduce some definitions and the p-inverse distribution. In Section 3, we present our algorithm for F_2. In Section 4, we present our generic framework for approximating F_p as well as our main result. In Section A of the full version of this paper, we present the detailed construction of each subroutine used in Section 4, and the details of the main algorithm. In Section 6, we introduce our deterministic algorithm, which uses our randomized algorithm as a subroutine. We also show its optimality in the same section.

2 Preliminaries

▶ **Definition 1** (Aggregate Streaming). An *insertion-only stream* (or simply *stream*) $\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is a sequence of integers, where each $a_i \in [n]$. A *weighted stream* $\mathcal{S}' = \langle (a_1, w_1), (a_2, w_2), \ldots, (a_m, w_m) \rangle$ is a sequence of pairs, where each $a_i \in [n]$ and each $w_m \in \mathbb{R}$. The insertion-only stream $\mathcal{S}$ is a special case of a weighted stream with all the weights being 1. The frequency $f_i(\mathcal{S}')$ of a weighted stream is defined as the sum of all weights of the item i. Formally,

$$f_i(\mathcal{S}') := \sum_{j=1}^{m} \mathbb{I}(a_j = i) w_j.$$

The frequency vector $V(\mathcal{S}') \in \mathbb{R}^n$ is the vector with i-th coordinate equal to $f_i(\mathcal{S}')$, for each $i \in [n]$. The p-th frequency moment, for $p \geq 0$, is defined as

$$F_p(\mathcal{S}') := \|V(\mathcal{S}')\|_p^p := \sum_{i=1}^{n} f_i^p(\mathcal{S}').$$

For time $0 < t_1 \leq t_2 \leq m$, we let

$$\mathcal{S}'^{:t_1} := \langle (a_1, w_1), (a_2, w_2), \ldots, (a_{t_1}, w_{t_1}) \rangle \quad \text{and}$$
$$\mathcal{S}'^{t_1:t_2} := \langle (a_{t_1}, w_{t_1}), (a_{t_1+1}, w_{t_1+1}), \ldots, (a_{t_2}, w_{t_2}) \rangle$$

be sub-streams of $\mathcal{S}'$ from 1 to t_1 or from t_1 to t_2.

We introduce a discretized version of the α-Fréchet distribution: the α-Inverse distribution.

▶ **Definition 2.** Fixing an $\alpha > 0$, we say a random variable X on $\mathbb{N}$ is drawn from an α-*Inverse* distribution if

$$\mathbb{P}[X < x] = 1 - \tfrac{1}{x^\alpha} \quad \text{for} \quad x \in \mathbb{N}_+.$$

▶ **Definition 3** (α-Scaling Transformation)**.** Given an $\alpha > 0$, an *α-scaling transformation (ST)* $\mathcal{T}_{k,\alpha} : [n]^m \to ([k] \times [n] \times \mathbb{R})^m$ is a function acting on a stream of length m on the universe $[n]$. On input stream $\mathcal{S}$, it outputs a weighted stream $\mathcal{S}'$ of length km on universe $[k] \times [n]$ via the following operation: let $X_{i,j}$ be identically distributed and independent (or, actually, limited independence suffices) α-Inverse random variables, where $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, n$. For each $a \in \mathcal{S}$, the transformation outputs

$$\mathcal{T}_{k,\alpha}(a) \to ((a, 1), X_{1,a}), ((a, 2), X_{2,a}), \ldots, ((a, k), X_{k,a}).$$

The next lemma shows that the $\frac{k}{2}$-th largest element of the transformed frequency vector gives a good approximation to the α-norm of the vector.

▶ **Lemma 4.** *Let $\mathcal{S}$ be a stream of items from the universe $[n]$. Let $\mathcal{T}_{k,\alpha}$ be a pairwise independent α-ST with $k \geq \frac{160}{\alpha^2\epsilon^2}$ being an even integer and $\alpha \geq 0$, where $\epsilon \in (0, \frac{1}{2\alpha})$. Let $\mathcal{S}' = T_{k,\alpha}(\mathcal{S})$. Define the two sets,*

$$U_+ := \{(a, r) \in [n] \times [k] : f_{(a,r)}(\mathcal{S}') \geq 2^{1/\alpha}(1 - \epsilon)\|V(\mathcal{S})\|_\alpha\}$$

and

$$U_- := \{(a, r) \in [n] \times [k] : f_{(a,r)}(\mathcal{S}') < 2^{1/\alpha}(1 + \epsilon)\|V(\mathcal{S})\|_\alpha\}.$$

Then with probability at least 0.9*,*

$$|U_+| \geq \tfrac{k}{2}, \quad |\overline{U_-}| < \tfrac{k}{2} \quad \textit{and} \quad |U_+ \cap U_-| > 0,$$

where $\overline{U}$ is the complement of the set U.

The proof of this lemma is provided in Section B of the full version.

3 A Simple F_2 Algorithm For Long Streams

We start with a very simple algorithm for approximating F_2 in a random order stream. We denote the algorithm by `RANDF2`. The algorithm has a parameter $b > 0$. It treats the stream updates as a series of length b blocks, i.e.,

$$\mathcal{S} = (B_1, B_2, \ldots, B_{m/b}).$$

Initialize a register $K = 0$. At any time, suppose the current block is B_i. The algorithm simply stores everything it sees until the end of B_i. After storing B_i entirely, the algorithm computes

$$K_i = \sum_{j=1} \binom{f_j(B_i)}{2}.$$

This can be done using $b \log n$ bits of space. Then, the counter K is updated as

$$K \leftarrow K + K_i.$$

At the end of the stream, the algorithm computes

$$Y = \frac{2K(m^2 - m)}{(b^2 - b)T} + m,$$

where T is the ID of the last complete block, and m is the length of the stream. The algorithm uses $O(b \log n)$ bits. By setting

$$b = \Theta[\max((\epsilon^2 \log n)^{-1}, 2) \cdot \log \tfrac{1}{\delta}],$$

we obtain the following theorem.

▶ **Theorem 5.** *Let $\mathcal{S}$ be a random order stream satisfying $F_2(\mathcal{S}) \geq m \cdot \log n$. After one pass over the stream $\mathcal{S}$, Y is a $(1 \pm \epsilon)$ approximation to $F_2(\mathcal{S})$ with probability at least $1 - \delta$. Moreover, to compute Y, `RANDF2` uses $\mathcal{O}(\epsilon^{-2} \log \delta^{-1} + \log n)$ bits of memory.*

Proof. For each $j \in [n]$, we let its stream updates be $u_1^{(j)}, u_2^{(j)}, \ldots, u_{f_j(\mathcal{S})}^{(j)}$. Then K_i is the number of pairs of the form $(u_{\ell_1}^{(j)}, u_{\ell_2}^{(j)})$ appearing in B_i. We denote $F_2 = F_2(\mathcal{S})$ and $F_1 = F_1(\mathcal{S})$ for simplicity. Thus,

$$\forall i \in [T] : \mathbb{E}(K_i) = \frac{\sum_{j=1}^{n} \binom{f_j(\mathcal{S})}{2}}{\binom{F_1}{2}} \binom{b}{2} = \frac{F_2 - m}{m^2 - m} \frac{(b^2 - b)}{2}.$$

Thus, by linearity of expectation,

$$\mathbb{E}[Y] = F_2.$$

We now prove that Y is concentrated around its mean. Notice that the algorithm can be viewed as sampling a number of "pairs". A pair is formed by two updates to the same universe element. There are $q = (F_2 - F_1)/2$ many pairs. Let $P = [q]$ denote the set of pairs. For each pair $z \in P$, we let X_z denote the indicator that X_z is sampled by some bucket. Let $K = \sum_{z \in P} X_z$. Note that this K is the same as the one denoted in the algorithm. Let $Q_z = \mathbb{E}[K | X_1, X_2, \ldots, X_z]$. The Q_z for $z = 1, 2, \ldots$ form a Doob martingale. Also notice that $|Q_z - Q_{z-1}| \leq 1$. Next, we proceed to bound the variance of

$$Q_z - Q_{z-1} = X_z | X_1, X_2, \ldots X_{z-1}.$$

For a pair $z \in P$, let a, b be the two nodes. Consider a fixed assignment of $X_1, X_2, \ldots X_{z-1}$. Also note that, knowing X_i, the two nodes of the i-th pair are assigned to some block.

Now, if from the $X_1, X_2, \ldots, X_{z-1}$, a, b are both assigned and to the same block, then $X_z = 1$ and otherwise $X_z = 0$. For both cases $\mathrm{Var}(Q_z - Q_{z-1}) = 0$. If a, b are assigned, but it cannot be determined if they are in the same block, then $\mathbb{P}[X_z = 1] \leq b/m$ and thus $\mathrm{Var}(X_z) \leq b/m$. If only one of a, b is assigned, then $\mathbb{P}[X_z = 1] \leq b/m$, and thus

$\mathrm{Var}(X_z) \le b/m$. Lastly, if both a, b are not assigned, then $\mathbb{P}[X_z = 1] \le b/m$. Thus $\mathrm{Var}(X_z) \le b/m$. Overall, we have that $v_z^2 := \mathrm{Var}(Q_z - Q_{z-1}) \le b/m$ for all possible $X_1, X_2, \ldots, X_{z-1}$. Let

$$V = \sum_z v_z^2 \le \tfrac{b(F_2-m)}{2m}.$$

Next, by Bernstein's inequality [14], we have that,

$$\mathbb{P}[|K - \mathbb{E}(K)| \ge t] \le 2\exp\left(-\tfrac{t^2}{2V(1+t/3V)}\right).$$

Since we need to have a $(1 \pm \epsilon)$ approximation to F_2, we can set

$$\frac{2t(m^2-m)b}{(b^2-b)m} \le \epsilon F_2 \quad \text{and} \quad t = \epsilon\frac{F_2 b}{2m}.$$

Since $F_2 \ge m \cdot \log n \cdot \log \frac{1}{\delta}$, and ϵ is sufficiently small, we can bound the error probability by:

$$\begin{aligned}\mathbb{P}[|K - \mathbb{E}(K)| \ge t] &\le 2\exp\left(-\frac{t^2}{2V(1+t/3V)}\right)\\ &\le 2\exp\left(-\frac{\epsilon^2 F_2^2 b}{8m(F_2-m)}\right) \le \delta, \end{aligned} \tag{1}$$

for $b = \Omega(\frac{\log \frac{1}{\delta}}{\epsilon^2 \cdot \log n})$. Finally, since $K \in \mathbb{E}(K) \pm \epsilon F_2 b/(2m)$, we have

$$Y \in F_2 \pm \epsilon\tfrac{F_2 b}{2m} \cdot \tfrac{2(m^2-m)b}{(b^2-b)m} \subset (1 \pm \epsilon)F_2$$

as desired. ◀

4 A Generic Framework for F_p Estimation

In this section, we first construct a generic framework for F_p estimation, and then we construct all the components in subsequent sections. For a random order stream $\mathcal{S}$, we will need the following three components to construct an F_p estimation algorithm.

- A counter that stores the time for the current update;
- An algorithm that gives a constant approximation to the current $F_p(\mathcal{S}^{:t})$;
- An algorithm that computes an accurate $(1 \pm \epsilon)$ approximation to $F_p(\mathcal{S})$ given $\mathrm{poly}(\log n, \epsilon^{-1})$ approximations to m and $F_p(\mathcal{S})$.

To begin, we denote by `C2Fp` a data structure that, once initialized with a $\mathrm{poly}(\log n, \epsilon^{-1})$-approximation of both the length and p-th frequency moments, $F_p(\mathcal{S})$, supports two operations: update and query. At the end of the stream, `C2Fp`.query() returns either `Fail` or a $(1 \pm \epsilon)$-approximation to F_p of the input stream $\mathcal{S}$. Component (1) will be used to guess the length of the stream, and component (2) will be used to guess an approximation to $F_p(\mathcal{S})$. We denote component (2) by `ConstFp`, which is a data structure that supports both update and query operations. `ConstFp`.query() returns a 2-approximation to $F_p(\mathcal{S}^{:t})$ at some fixed t. The full framework is described in Algorithm 1.

Our full algorithm is denoted by `RndF`$_p$, which uses `C2Fp` as a subroutine. From a high level, the algorithm constantly guesses the length of the stream. If at some point in time the algorithm finds that the current guess of the length is at least a factor $C = \mathrm{poly}(\epsilon^{-1}, \log n)$ smaller than the true value of the stream, then the algorithm initializes a new instance of `C2Fp` to estimate the F_p value of the stream. At the end, it is guaranteed that a stored

Algorithm 1: Full algorithm for F_p in random order: $\texttt{RndF}_p$.

Data:
$\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is a random order stream of length m from a universe $[n]$ (known in advance);
$p \in (0, 2]$ is a real number, which is a constant;

Initialize (p, n, ϵ, δ)**:**
  $m_0 \leftarrow 1$, $m_1 \leftarrow 1$ $G_0 \leftarrow 1$. Here m_0 is the approximate length, G_0 is a guess of F_p, ϵ is the target precision, and δ is the failure probability;
  $A_1 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$, $A_2 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$;
  $A_3 \leftarrow$new $\texttt{ConstFp}(p, n, \delta/3)$;
  $C \leftarrow \text{poly}(\frac{1}{\epsilon}, \log \frac{n}{\delta})$;

Update a**:**
  A_1.update(a), A_2.update(a), A_3.update(a);
  $m_1 \leftarrow m_1 + 1$;
  if $m_1 \geq C m_0$ **then**
    $A_1 \leftarrow A_2$;
    $G_0 \leftarrow A_3$.query();
    $m_0 \leftarrow m_1$;
    $A_2 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$;

Query():
  return A_1.query();

instance of C2Fp uses at least a $(1 - \text{poly}(\epsilon, 1/\log n))$ portion of the stream to approximate the frequency moments. It can be verified that an accurate estimation of F_p of this portion of the stream will serve as a good estimator for the overall stream. Therefore, if C2Fp is able to output the correct answer with high probability, then the algorithm RndF is guaranteed to be correct with high probability.

▶ **Theorem 6** (Main Theorem). *For fixed $p \in [0, 2)$, $\epsilon \in (0, 1)$,$\delta \in (\frac{1}{\text{poly}(n)}, \frac{1}{2})$, and $n \in \mathbb{N}_+$, algorithm $\texttt{RndF}_p$, makes a single pass over a random order stream $\mathcal{S}$ on universe $[n]$, and outputs a number $\hat{F}$ such that, with probability at least $1 - \delta$,*

$$(1 - \epsilon)F_p(\mathcal{S}) \leq \hat{F} \leq (1 + \epsilon)F_p(\mathcal{S}),$$

where the probability is over both the randomness of the algorithm and the randomness of the data stream. The algorithm uses

$$\mathcal{O}[(\frac{1}{\epsilon^2}(\log\log n + \log\frac{1}{\epsilon})^4 + \log n)\log\frac{1}{\delta}]$$

bits of memory in the worst case.

Proof. Without loss of generality, we assume $F_p(\mathcal{S}) = \Omega(\text{poly}\frac{1}{\epsilon}\text{poly}\log\frac{n}{\delta})$, since otherwise we can use a turnstile F_p algorithm with memory $\mathcal{O}\big(\frac{1}{\epsilon^2}\log\log\frac{n}{\delta} + \frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\big)$ bits to solve the F_p estimation problem. Initialize $m_0 = 1$. Let $\mathcal{S}' = \mathcal{S}^{m_0:m}$. By definition of the algorithm, A_1 is an instance of C2Fp that runs on $\mathcal{S}'$. Let $\mathcal{S}'' = \mathcal{S}^{0:m_0}$ and $C = \text{poly}\epsilon^{-1}\text{polylog}\frac{n}{\delta}$. By definition of the algorithm, we always update m_0 so that

$$\frac{m}{C^2} \leq m_0 \leq \frac{m}{C},$$

at the end of the stream. By Lemma 24 and Lemma 25 of the full version of this paper, we have, with probability at least $1 - \delta/3$,

$$\frac{F_p(\mathcal{S})}{5^p C^{2p}} \leq F_p(\mathcal{S}'') \leq (17\log\frac{20n}{\delta})^p(C^{-1}F_p(\mathcal{S}) + 4\log\frac{20n}{\delta}) \leq \frac{(18\log\frac{20n}{\delta})^p}{C}F_p(\mathcal{S}),$$

where the last inequality holds for sufficiently large $F_p(\mathcal{S})$. Conditioned on this event, we obtain that

$$\|V(\mathcal{S}'')\|_p \leq \frac{(18 \log \frac{20n}{\delta})}{C^{1/p}} \|V(\mathcal{S})\|_p \leq \frac{\epsilon}{3p} \|V(\mathcal{S})\|_p,$$

for sufficiently large C. By the triangle inequality, we obtain

$$(1 - \epsilon/(3p))\|V(\mathcal{S})\|_p \leq \|V(\mathcal{S})\|_p - \|V(\mathcal{S}'')\|_p \leq \|V(\mathcal{S}')\|_p \leq \|V(\mathcal{S})\|_p.$$

Thus $F_p(\mathcal{S}')$ is a $(1 \pm \epsilon/3)$ approximation to $F_p(\mathcal{S})$.

In the algorithm, A_3 is an instance of `ConstFp`, i.e., by Theorem 7. Let G_0 be the output of A_3.query(). Since with probability at least $1 - \delta/3$, A_3 outputs a c_0 approximation to $F_p(\mathcal{S}'')$ for some constant c_0, we obtain that G_0 is a $5^p c_0 C^{2p} (\log \frac{20n}{\delta})^{p-1}$ approximation to $F_p(\mathcal{S}')$.

In the algorithm, A_1 is an instance of `C2Fp`, which runs on the stream $\mathcal{S}'$ at any time t with the required parameters, i.e., G_0. By Theorem 8, with probability at least $1 - \delta/3$, A_1.query() outputs a $(1 \pm \epsilon/3)$ approximation to $F_p(\mathcal{S}')$, and thus a $(1 \pm \epsilon)$ approximation to $F_p(\mathcal{S})$. By a union bound, the overall algorithm is correct with probability at least $1 - \delta$.

By Theorem 16 and Theorem 17, of the full version, the space needed for A_1 and A_2 is

$$\mathcal{O}[(\frac{1}{\epsilon^2}(\log \log n + \log \frac{1}{\epsilon})^4 + \log n) \log \frac{1}{\delta}].$$

The space needed for A_3 is $\mathcal{O}(\log n \log \frac{1}{\delta})$ (Theorem 7). Thus the total space is dominated by the total space used by A_1 and A_2 as desired. ◀

The following is a theorem required in the above proof.

▶ **Theorem 7** (Const. F_p approx., [21]).
For a fixed n, there exists a turnstile streaming algorithm, which on input a stream $\mathcal{S}$ of length m, outputs a number $F \in (1 \pm \epsilon)F_p(\mathcal{S})$ with probability at least $1 - \delta$. The algorithm uses $\mathcal{O}(\epsilon^{-2} \log m + \log \log(n)) \log \delta^{-1})$ bits of space in the worst case.

In subsequent sections, we will construct the `C2Fp` Algorithm.

5 A $(1 \pm \epsilon)$ Approximation to F_p With a Prior

In this section, we construct the algorithm `C2Fp`. We assume that the input is a random order stream and that the algorithm is given two parameters, $\hat{m}$ and G, which are $\text{poly}(\epsilon^{-1}, \log n)$ approximations to the length and the p-th frequency moments of the stream $\mathcal{S}$, respectively.

5.1 High Level Idea

Although the high level idea is introduced in the introduction, we repeat it here with more details for better understanding of the algorithm. To illustrate the intuition of the algorithm, we first consider a constant factor approximation algorithm. Estimating the frequency moments can be reduced to finding the *heavy hitters* of a scaled vector, as shown in Lemma 4. Suppose the frequency vector in a stream is $v = (f_1(\mathcal{S}), f_2(\mathcal{S}), \ldots, f_n(\mathcal{S}))$, and the scaling applied to it is $X = (X_1, X_2, \ldots, X_n)$, where the X_i are pairwise independent *p-Inverse* (see Definition 2) random variables. Let i^* be the maximum of the scaled vector. By Lemma 4, we expect $X_{i^*}^p f_{i^*}^p(\mathcal{S}) \approx F_p$. We group the coordinates of v into $\Theta(\log n)$ levels by their scalings, i.e., if $2^{-w} F_p < X_i^p \leq 2^{-w+1} F_p$, then i is in level w. Let $Z_w \subset [n]$ be the universe items in

level w. We observe that if $i^* \in Z_w$, then $f_{i^*}^p(\mathcal{S}) = \Omega(2^w)$. Luckily, we can also show that, in expectation, i^* is an F_p-heavy hitter of the substream induced by Z_w. Our algorithm is simply *looking for i^* from Z_w for every $w \in [\log n]$*. One may notice that if we run the search instances in parallel, then there will be a $\log n$ factor blowup in the space. However, we can show that in a random order stream, one can choose a $w_0 = \Theta(\log \log n)$ such that

1. for all $w > w_0$: the search for i^* can be done in one pass and in series for each w.
2. for all $w \le w_0$: with high probability, $|Z_w| = \operatorname{poly} \log n$. We thus do a brute force search for each level w below w_0 in parallel.

The final space overhead is a $\operatorname{poly}(w_0)$ factor rather than $\Theta(\log n)$.

To reduce the error from constant to $(1 \pm \epsilon)$, we repeat the above process $\Theta\left(\frac{1}{\epsilon^2}\right)$ times conceptually. Namely, we apply a p-ST $\mathcal{T}_{k,p}$ transformation to the stream, where $k = \Theta\left(\frac{1}{\epsilon^2}\right)$. For $r = 1, 2, \ldots, [k]$, $i = 1, 2, \ldots, n$, we denote the scaling p-Inverse random variable by $X_i^{(r)}$. We wish to find the heavy hitter for each r using the same procedure described above. By Lemma 4, the $k/2$-th largest of all the outputs serves as a good approximation to $F_p(\mathcal{S})$.

5.2 The Algorithm

The algorithm needs three components, `SmallApprox`, `SmallCont` and `LargeCont`. All these algorithms support "update" and "query" operations. `SmallApprox` returns fail if $F_p(\mathcal{S})$ is much larger than $\operatorname{poly}(\epsilon^{-1}, \log n)$, otherwise returns an approximation to $F_p(\mathcal{S})$. `SmallApprox` is a turnstile streaming F_p algorithm [21] but with restricted memory. Once the memory exceeds the memory quota, the algorithm simply returns `Fail`. `SmallCont` estimates the contribution from the small-valued frequencies and `LargeCont` estimates the contribution from the large-valued frequencies. The correctness of these algorithms is presented in Theorem 16 and 17 of the full version of this paper. The full algorithm is presented in Algorithm 2. The following theorem guarantees its correctness.

▶ **Theorem 8.** *Fix $p \in [0, 2]$, $\epsilon \in (0, 1/2)$ and $\delta = \Omega(1/\operatorname{poly}(n))$. Let $\mathcal{S}$ be a random order stream on universe $[n]$ and with length m. Given that $C_0^{-1} F_p(\mathcal{S}) \le G_0 \le F_p(\mathcal{S})$ and $C_0^{-1} m \le m_0 \le m$ for some $C_0 = \operatorname{poly}(\epsilon^{-1}, \log n)$, there exists an algorithm $\mathcal{A}$, which makes a single pass over $\mathcal{S}$ and outputs a number F such that $F \in (1 \pm \epsilon) F_p(\mathcal{S})$ with probability at least $1 - \delta$, where the probability is over both the randomness of the algorithm and of the order of the stream. The algorithm uses*

$$\mathcal{O}[(\tfrac{1}{\epsilon^2}(\log \log n + \log \tfrac{1}{\epsilon})^4 + \log n) \log \tfrac{1}{\delta}]$$

bits of memory in the worst case.

We postpone the full proof and detailed algorithmic constructions to the appendix.

6 Deterministic Algorithm for F_p Approximation

In this section we introduce our deterministic algorithm for F_p approximation, which follows from our randomized algorithm with an initial space-efficient randomness extraction procedure applied to a prefix of the stream.

6.1 Upper Bound

▶ **Theorem 9.** *Fix $p \ge 0, \epsilon \in (0, 1)$. There exists a deterministic algorithm that makes a single pass over a random order stream $\mathcal{S}$ on the universe $[n]$, and outputs a number*

Algorithm 2: F_p-Algorithm with Approximation: `C2Fp`$(p, \epsilon, n, m_0, G_0)$.

Data:
$p \in [0, 2]$, a real number;
$L \in [\frac{F_p(\mathcal{S})}{C_0}, F_p(\mathcal{S})]$ for some $C_0 = \Theta[\text{poly}(\epsilon^{-1}, \log n)]$;
$m_0 \in [\frac{m}{C_0}, m]$;
$\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is random order stream of length m;
Result: $F \in (1 \pm \epsilon)F_p(\mathcal{S})$;

Initialize(p, n, ϵ, δ)**:**
  $k \leftarrow \Theta\big(\frac{1}{\epsilon^2}\big)$;
  $X_i^{(r)} \sim p$-Inverse distribution for $i \in [n]$ and $r \in [k]$, pairwise independent;
  $w_0 \leftarrow d_0\big(\log\log n + \log\frac{1}{\epsilon}\big)$ for some large constant d_0;
  Let $K \in \mathbb{R}^{n \times k}$ have entries $K_{i,r} = X_i^{(r)} v_i$ (only for notational purposes);
  $B_1 \leftarrow$ new `SmallApprox`(p, n, ϵ);
  $B_2 \leftarrow$ new `SmallCont`$(p, n, k, \epsilon, w_0, L, \{X_i^r\})$;
  $B_3 \leftarrow$ new `LargeCont`$(p, n, k, \epsilon, w_0, L, \{X_i^r\})$;
Update(a)**:**
  B_1.update(a); B_2.update(a); B_3.update(a);
Query:
  if B_1.*query*$() \neq$`Fail` **then**
    return B_1.query()
  else if B_2.*query*$() =$`Fail` *or* B_3.*query*$() =$`Fail` **then**
    return `Fail`;
  else
    $R \leftarrow$ the $(k/2)$-th largest element of B_2.query() $\circ$ B_3.query();
    return $(R)^p/2$

$F \in (1 \pm \epsilon)F_p(\mathcal{S})$ with probability at least $1 - \delta$, where the randomness is over the order of the stream updates. The algorithm uses

$$\mathcal{O}\left[\frac{1}{\epsilon^2}\left(\log\log n + \log\frac{1}{\epsilon}\right)^4 \log\frac{1}{\delta} + \log n \cdot \left(\log\log n + \log\frac{1}{\delta}\right) \cdot \frac{1}{\delta} + \log n \log\frac{1}{\epsilon}\right]$$

bits of memory, provided $\delta \geq 1/\text{poly}(n)$.

Proof. W.l.o.g., we assume $\epsilon \geq 1/\sqrt{n}$, since otherwise we can simply store an approximate counter for each item in the stream. It is sufficient to show that we are able to derandomize the randomized algorithm using the random updates from the stream using a near-logarithmic number of bits of space. First we pick $s = \mathcal{O}(\log\log n + \log\delta^{-1})$ and store all the universe items, their frequencies and their first arrival times until we obtain s distinct items in the stream. Let z_1 denote when the this happens. We assume the stream is long enough that this step can be done, since otherwise we obtain an exact estimate of F_p. We show in Section 6.1.1 how to obtain a nearly uniformly random seed of s bits.

We thus obtain a nearly uniform sample from all the prime numbers with $\mathcal{O}(\log\log n + \log\frac{1}{\delta})$ bits (note that there are $\text{poly}(\log n/\delta)$ many such prime numbers). Let this sampled prime number be q. For the next $\mathcal{O}(\log n/\delta)$ distinct universe items, denoted by R, we argue that with probability at least $1 - \delta$, all of them are distinct modulo q. Indeed, consider any $r_1, r_2 \in R$ with $r_1 \neq r_2$. Then $r_1 - r_2$ can have at most $\log n$ prime factors ([17], p.355). For

all $\binom{|R|}{2}$ pairs, their differences can have at most $\mathcal{O}(\log^3 n/\delta^3)$ distinct prime factors in total. Thus with probability at least $1-\delta$, q does not divide any of the differences. Therefore the set R is mapped to distinct numbers modulo q. The value q can be stored using at $\mathcal{O}(\log\log n + \log\frac{1}{\delta})$ bits.

Next we approximately store the frequencies of each item in R using the random order stream. To do this, we first fix the following numbers $g_1 = 1, g_2 = 2, g_3 = 4, \ldots, g_i = 2^{i-1}$. For each $r \in R$, we store the largest number i such that $f_r(\mathcal{S}^{z_1:g_i}) = \text{poly}(\log n, \epsilon^{-1})$ and $f_r(\mathcal{S}^{z_1:g_i})$ as well. Therefore, such an operation only costs $\mathcal{O}(\frac{\log n}{\delta}(\log\log n + \log\frac{1}{\delta} + \log\epsilon^{-1}))$ bits. By Lemma 22 of the full version, the frequency of each item is preserved up to a $(1\pm\epsilon)$ factor with probability at least $1 - 1/\text{poly}(n)$. Note that if the stream ends before we observe all of R, we obtain a good approximation to $F_p(\mathcal{S})$ immediately. We also store the first occurrence in the stream of each item in R. We also store the parity of the first appearance of each item.

Repeating the extraction argument in Section 6.1.1 for the set R, we can now extract $\mathcal{O}(\log n)$ bits that is $(1\pm\delta)$ close to uniform. Given these bits, it now suffices to run our earlier randomized algorithm on the remaining part of the stream.

There is one last problem remaining, however. Namely, it may be the case that the stream used for extracting random bits contributes too much to $F_p(\mathcal{S})$, causing the estimation of the randomized algorithm to have too much error (since the prefix and the suffix of the stream share the same items, we need to take the p-th power of the sum of their frequencies). This problem can be solved as follows – we can continue the frequency estimation in parallel with the randomized algorithm until the F_p value becomes at least a $1/\epsilon$ factor larger than the time when we initialized our randomized algorithm. Therefore, if the stream ends before this happens, then we use the frequency estimates for calculating $F_p(\mathcal{S})$ from our deterministic algorithm. Otherwise we use the value of the randomized algorithm (which is seeded with the seed found by our deterministic algorithm). In either case, the overall error is at most ϵF_p. ◀

6.1.1 Derandomization

Let $s > 0$ be a parameter. Suppose we store $t = \mathcal{O}(s/\delta)$ distinct universe items with their approximate frequencies as well as the IDs and the parities of their first appearances in the stream. Denote the set of these items by H and the overall length of the stream as m'. First, we sort the items by their approximate counts and take the smallest $\delta t/100$ items as set L. We additionally sort the items in L by their IDs, and obtain a bit string b of length $|L|$, where each bit b_i is the parity of the first appearance of the i-th item in L. Since L contains the smallest $\delta t/100$ items of H, we have for each $w \in L$, $f_w(\mathcal{S}^{0:m'}) \le m'/t/(1-\delta/100)$. Thus for each bit b_i,

$$\mathbb{P}[b_i = 0], \mathbb{P}[b_i = 1] \in \frac{1}{2} \pm \frac{2}{m'}.$$

and

$$\forall x \in \{0,1\}^{|L|} : \mathbb{P}[b = x] \in (\frac{1}{2} \pm \frac{2}{m'})^{|L|} \subset \frac{1}{2^{|L|}}(1 \pm \frac{5|L|}{m'}) \subset \frac{1}{2^{|L|}}(1 \pm \frac{\delta}{20}).$$

As such, we obtain a bit stream of length $\Omega(s)$, that is close to uniform bits up to a $(1\pm\delta)$ factor.

References

1. Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 10–20. ACM, 1999.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.
3. Alexandr Andoni. High frequency moment via max stability. *Unpublished manuscript*, 2012.
4. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 363–372. IEEE, 2011.
5. Alexandr Andoni, Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. Better bounds for frequency moments in random-order streams. *arXiv preprint arXiv:0808.2222*, 2008.
6. Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 740–753. ACM, 2016.
7. Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o (nˆ(1-2/k)) bits. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 28. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.
8. Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *arXiv preprint arXiv:1011.2571*, 2010.
9. Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 641–650. ACM, 2008.
10. Amit Chakrabarti, TS Jayram, and Mihai Patrascu. Tight lower bounds for selection in randomly ordered streams. In *SODA*, pages 720–729, 2008.
11. Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 107–117. IEEE, 2003.
12. Michael Crouch, Andrew McGregor, Gregory Valiant, and David P Woodruff. Stochastic streams: Sample complexity vs. space complexity. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
13. Erik Demaine, Alejandro López-Ortiz, and J Munro. Frequency estimation of internet packet streams with limited space. *Algorithms—ESA 2002*, pages 11–20, 2002.
14. Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
15. Sudipto Guha and Zhiyi Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. *Automata, Languages and Programming*, pages 513–524, 2009.
16. Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.
17. Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers*. Oxford University Press, 1979.
18. Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 189–197. IEEE, 2000.
19. Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

20 Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 745–754. ACM, 2011.

21 Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

22 J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

23 David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 167–175. Society for Industrial and Applied Mathematics, 2004.