# An Optimal Algorithm for the Distinct Elements Problem


Daniel M. Kane *
Harvard University
One Oxford Street
Cambridge, MA 02138
dankane@math.harvard.edu

Jelani Nelson †
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
minilek@mit.edu

David P. Woodruff
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
dpwoodru@us.ibm.com



## ABSTRACT

We give the first optimal algorithm for estimating the number of distinct elements in a data stream, closing a long line of theoretical research on this problem begun by Flajolet and Martin in their seminal paper in FOCS 1983. This problem has applications to query optimization, Internet routing, network topology, and data mining. For a stream of indices in $\{1,\ldots,n\}$, our algorithm computes a $(1\pm\varepsilon)$-approximation using an optimal $O(\varepsilon^{-2}+\log(n))$ bits of space with 2/3 success probability, where $0<\varepsilon<1$ is given. This probability can be amplified by independent repetition. Furthermore, our algorithm processes each stream update in $O(1)$ worst-case time, and can report an estimate at any point midstream in $O(1)$ worst-case time, thus settling both the space and time complexities simultaneously.

We also give an algorithm to estimate the Hamming norm of a stream, a generalization of the number of distinct elements, which is useful in data cleaning, packet tracing, and database auditing. Our algorithm uses nearly optimal space, and has optimal $O(1)$ update and reporting times.



**Categories and Subject Descriptors:** F.2.0 [Analysis of Algorithms and Problem Complexity]: General; H.2.m [Database Management]: Miscellaneous

**General Terms:** Algorithms, Theory

**Keywords:** distinct elements, streaming, query optimization, data mining


## 1. INTRODUCTION

Estimating the number of distinct elements in a data stream is a fundamental problem in network traffic monitoring, query optimization, data mining, and several other database areas. For example, this statistic is useful for selecting a minimum-cost query plan [33], database design [18], OLAP [30, 34], data integration [10, 14], and data warehousing [1].

In network traffic monitoring, routers with limited memory track statistics such as distinct destination IPs, requested URLs, and source-destination pairs on a link. Distinct elements estimation is also useful in detecting Denial of Service attacks and port scans [2, 17]. In such applications the data is too large to fit at once in main memory or too massive to be stored, being a continuous flow of data packets. This makes small-space algorithms necessary. Furthermore, the algorithm should process each stream update (i.e., packet) quickly to keep up with network speeds. For example, Estan *et al* [17] reported packet header information being produced at .5GB per hour while estimating the spread of the Code Red worm, for which they needed to estimate the number of distinct Code Red sources passing through a link.

Yet another application is to data mining: for example, estimating the number of distinct queries made to a search engine, or distinct users clicking on a link or visiting a website. Distinct item estimation was also used in estimating connectivity properties of the Internet graph [32].

We formally model the problem as follows. We see a stream $i_1,\ldots,i_m$ of indices $i_j\in[n]$, and our goal is to compute $F_0=|\{i_1,\ldots,i_m\}|$, the number of distinct indices that appeared, using as little space as possible. Since it is known that exact or deterministic computation of $F_0$ requires linear space [3], we settle for computing a value $\widetilde{F}_0\in[(1-\varepsilon)F_0,(1+\varepsilon)F_0]$ for some given $0<\varepsilon<1$ with probability 2/3, over the randomness used by the algorithm. This probability can be amplified by independent repetition.

The problem of space-efficient $F_0$-estimation is well-studied, beginning with the work of Flajolet and Martin [20], and continuing with a long line of research, [3, 4, 5, 6, 9, 12, 16, 17, 19, 23, 24, 26, 36]. In this work, we finally settle both the space- and time-complexities of $F_0$-estimation by giving an algorithm using $O(\varepsilon^{-2}+\log(n))$ bits of space, with worst-case update and reporting times $O(1)$. By update time, we mean the time to process a stream token, and by reporting time, we mean the time to output an estimate of $F_0$ at any point in the stream. Our space upper bound matches the known lower bounds [3, 26, 36] up to a constant factor, and the $O(1)$ update and reporting times are clearly optimal. A detailed comparison of our results to those in previous work is given in Figure 1. There is a wide spectrum of time/space tradeoffs but the key points are that none of the previous


*Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

†Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship, and in part by the Center for Massive Data Algorithmics (MADALGO) - a center of the Danish National Research Foundation. Part of this work was done while the author was at the IBM Almaden Research Center.




*PODS'10,* June 6–11, 2010, Indianapolis, Indiana, USA.

| Paper | Space | Update Time | Notes |
|---|---|---|---|
| [20] | $O(\log n)$ | - | Assumes random oracle, constant $\varepsilon$ |
| [3] | $O(\log n)$ | $O(\log n)$ | Only works for constant $\varepsilon$ |
| [24] | $O(\varepsilon^{-2} \log n)$ | $O(\varepsilon^{-2})$ | |
| [5] | $O(\varepsilon^{-3} \log n)$ | $O(\varepsilon^{-3})$ | |
| [4] | $O(\varepsilon^{-2} \log n)$ | $O(\log(\varepsilon^{-1}))$ | Algorithm I in the paper |
| [4] | $O(\varepsilon^{-2} \log\log n + \mathrm{poly}(\log(\varepsilon^{-1}), \log\log n) \log n)$ | $\varepsilon^{-2}\,\mathrm{poly}(\log\log n + \log(\varepsilon^{-1}))$ | Algorithm II in the paper |
| [4] | $O(\varepsilon^{-2}(\log(\varepsilon^{-1}) + \log\log n) + \log n)$ | $O(\varepsilon^{-2}(\log(\varepsilon^{-1}) + \log\log n))$ | Algorithm III in the paper |
| [16] | $O(\varepsilon^{-2} \log\log n + \log n)$ | - | Assumes random oracle, additive error |
| [17] | $O(\varepsilon^{-2} \log n)$ | - | Assumes random oracle |
| [6] | $O(\varepsilon^{-2} \log n)$ | $O(\log(\varepsilon^{-1}))$ | |
| [19] | $O(\varepsilon^{-2} \log\log n + \log n)$ | - | Assumes random oracle, additive error |
| This work | $O(\varepsilon^{-2} + \log n)$ | $O(1)$ | Optimal |

**Figure 1: Comparison of our algorithm to previous algorithms on estimating the number of distinct elements in a data stream.**

algorithms achieved our optimal $O(\varepsilon^{-2}+\log n)$ bits of space, and the only ones to achieve optimal $O(1)$ update and/or reporting time had various restrictions, e.g., the assumption of access to a random oracle (that is, a truly random hash function) and/or a small constant additive error in the estimate. The best previous algorithms without any assumptions are due to Bar Yossef *et al* [4], who provide algorithms with various tradeoffs (Algorithms I, II, and III in Figure 1).

We also give a new algorithm for estimating $L_0$, also known as the *Hamming norm* of a vector [13], with optimal running times and near-optimal space. This problem is a generalization of $F_0$-estimation to the case when items can be removed from the stream. While $F_0$-estimation is useful for a single stream or for taking unions of streams if there are no deletions, $L_0$-estimation can be applied to a pair of streams to measure the number of unequal item counts. This makes it more flexible than $F_0$, and can be used in applications such as maintaining ad-hoc communication networks amongst cheap sensors [25]. It also has applications to data cleaning to find columns that are mostly similar [14]. Even if the rows in the two columns are in different orders, streaming algorithms for $L_0$ can quickly identify similar columns. As with $F_0$, $L_0$-estimation is also useful for packet tracing and database auditing [13].

Formally, in this problem there is a vector $x = (x_1, \ldots, x_n)$ which starts off as the 0 vector, and receives $m$ updates of the form $(i, v) \in [n] \times \{-M, \ldots, M\}$ in a stream ($M$ is some positive integer). The update $(i, v)$ causes the change $x_i \leftarrow x_i + v$. At the end of the stream, we should output $(1 \pm \varepsilon)L_0$ with probability at least 2/3, where $L_0 = |\{i : x_i \neq 0\}|$. Note that $L_0$-estimation is a generalization of $F_0$-estimation, since in the latter case an index $i$ in the stream corresponds to the update $(i, 1)$ in an $L_0$-estimation problem.

We give an $L_0$-estimation algorithm with $O(1)$ update and reporting times, using $O(\varepsilon^{-2}\log(n)(\log(1/\varepsilon)+\log\log(mM)))$ bits of space, both of which improve upon the previously best known algorithm of Ganguly [22], which had $O(\log(1/\varepsilon))$ update time and required $O(\varepsilon^{-2}\log(n)\log(mM))$ space. Our update and reporting times are optimal, and the space is optimal up to the $\log(1/\varepsilon) + \log\log(mM)$ term due to known lower bounds [3, 27]. Furthermore, unlike with Ganguly's algorithm, our algorithm does not require that $x_i \geq 0$ for each $i$ to operate correctly.

## 1.1 Overview of our algorithms

Our algorithms build upon several techniques given in previous works, with added twists to achieve our stated performance. In for example [4], it was observed that if one somehow knows ahead of time a value $R = \Theta(F_0)$, refining to $(1 \pm \varepsilon)$-approximation becomes easier. For example, [4] suggested a "balls and bins" approach to estimating $F_0$ given such an $R$. The key intuition is that when hashing $A$ balls randomly into $K$ bins, the number of bins hit by at least one ball is highly concentrated about its expectation, and treating this expectation as a function of $A$ then inverting provides a good approximation to $A$ with high probability for $A = \Theta(K)$. Then if one subsamples each index in $[n]$ with probability $2^{-\log(R/K)}$, in expectation the number of distinct items surviving is $\Theta(K)$, at which point the balls-and-bins approach can be simulated by hashing indices (the "balls") into entries of a bitvector (the "bins").

Following the above scheme, an estimate of $F_0$ can be obtained by running a constant-factor approximation in parallel to obtain such an $R$ at the end of the stream, and meanwhile performing the above scheme for geometrically increasing guesses of $R$, one of which must be correct to within a constant factor. Thus, the bits tracked can be viewed as a bitmatrix: rows corresponding to $\log(n)$ levels of subsampling, and columns corresponding to entries in the bitvector. At the end of the stream, upon knowing $R$, the estimate from the appropriate level of subsampling is used. Such a scheme with $K = \Theta(1/\varepsilon^2)$ works, and gives $O(\varepsilon^{-2}\log(n))$ space, since there are $\log(n)$ levels of subsampling.

It was then first observed in [16] that, in fact, an estimator can be obtained without maintaining the full bitmatrix above. Specifically, for each column they gave an estimator that required only maintaining the deepest row with its bit set to 1. This allowed them to collapse the bitmatrix above to $O(\varepsilon^{-2}\log\log(n))$ bits. Though, their estimator and analysis required access to a purely random hash function.

Our $F_0$ algorithm is inspired by the above two algorithms of [4, 16]. We give a subroutine RoughEstimator using $O(\log(n))$ space which with high probability, simultaneously provides a constant-factor approximation to $F_0$ at all times in the stream. Previous subroutines gave a constant factor approximation to $F_0$ at any *particular* point in the stream with probability $1-\delta$ using $O(\log(n)\log(1/\delta))$ space; a good approximation at all times then required setting $\delta = 1/m$ to apply a union bound, thus requiring $O(\log(n)\log(m))$ space. The next observation is that if $R = \Theta(F_0)$, the largest row index with a 1 bit for any given column is heavily concentrated around the value $\log(F_0/K)$. Thus, if we *bitpack* the $K$ counters and store their offsets from $\log(R/K)$, we expect to only use $O(K)$ space for all counters combined. Whenever $R$ changes, we update all our offsets.

There are of course obvious obstacles in obtaining $O(1)$ running times, such as the occasional need to decrement all $K$ counters (when $R$ increases), or to locate the starting position of a counter in a bitpacked array when reading and writing entries. For the former, we use a "variable-bit-length array" data structure [7], and for the latter we use an approach inspired by the technique of deamortization of global rebuilding (see [29, Ch. 5]). Furthermore, we analyze our algorithm without assuming a truly random hash function, and show that a combination of fast $k$-wise independent hash functions [35] and uniform hashing [31] suffice to have sufficient concentration in all probabilistic events we consider.

Our $L_0$-estimation algorithm also uses subsampling and a balls-and-bins approach, but needs a different subroutine for obtaining the value $R$, and for representing the bitmatrix. Specifically, if one maintains each bit as a counter and tests for the counter being non-zero, frequencies of opposite sign may cancel to produce 0 and give false negatives. We instead store the dot product of frequencies in each counter with a random vector over a suitably large finite field. We remark that Ganguly's algorithm [22] is also based on a balls-and-bins approach, but on viewing the number of bins hit by *exactly* one ball (and not at least one ball), and the source of his algorithm's higher complexity stems from technical issues related to this difference.

## 1.2 Preliminaries

Throughout this paper, all space bounds are given in bits. We always use $m$ to denote stream length and $[n]$ to denote the universe (the notation $[n]$ represents $\{1,\ldots,n\}$). Without loss of generality, we assume $n$ is a power of 2, and $\varepsilon \le \varepsilon_0$ for some fixed constant $\varepsilon_0 > 0$. In the case of $L_0$, $M$ denotes an upper bound on the magnitude of updates to the $x_i$. We use the standard word RAM model, and running times are measured as the number of standard machine word operations (integer arithmetic, bitwise operations, and bitshifts). We assume a word size of at least $\Omega(\log(nmM))$ bits to be able to manipulate counters and indices in constant time.

For reals $A, B, \varepsilon \ge 0$, we use the notation $A = (1 \pm \varepsilon)B$ to denote that $A \in [(1-\varepsilon)B, (1+\varepsilon)B]$. We use $\mathrm{lsb}(x)$ to denote the (0-based index of) the least significant bit of a nonnegative integer $x$ when written in binary. For example, $\mathrm{lsb}(6) = 1$. We define $\mathrm{lsb}(0) = \log(n)$. All our logarithms are base 2 unless stated otherwise. We also use $\mathcal{H}_k(U,V)$ to denote some $k$-wise independent hash family of functions mapping $U$ into $V$. Using known constructions [11], a random $h \in \mathcal{H}_k(U,V)$ can be represented in $O(k\log(|U|+|V|))$ bits when $|U|, |V|$ are powers of 2, and computed in the same amount of space. Also, henceforth, whenever we discuss picking an $h \in \mathcal{H}_k(U,V)$, it should be understood that $h$ is being chosen as a random element of $\mathcal{H}_k(U,V)$.

When discussing $F_0$, for $t \in [m]$ we use $I(t)$ to denote $\{i_1,\ldots,i_t\}$, and define $F_0(t) = |I(t)|$. We sometimes use $I$ to denote $I(m)$ so that $F_0 = F_0(m) = |I|$. In the case of $L_0$-estimation, we use $I(t)$ to denote the $i$ with $x_i \neq 0$ at time $t$. For an algorithm which outputs an estimate $\widetilde{F}_0$ of $F_0$, we let $\widetilde{F}_0(t)$ be its estimate after only seeing the first $t$ updates (and similarly for $L_0$). More generally, for any variable $y$ kept as part of the internal state of any of our algorithms, we use $y(t)$ to denote the contents of that variable at time $t$.

Lastly, we analyze our algorithm without any idealized assumptions, such as access to a cryptographic hash function, or to a hash function which is truly random. Our analyses all take into account the space and time complexity required to store and compute the types of hash functions we use.

# 2. BALLS AND BINS WITH LIMITED INDEPENDENCE

In the analysis of the correctness of our algorithms, we require some understanding of the balls and bins random process with limited independence. We note that [4] also required a similar analysis, but was only concerned with approximately preserving the expectation under bounded independence whereas we are also concerned with approximately preserving the variance. Specifically, consider throwing a set of $A$ balls into $K$ bins at random and wishing to understand the random variable $X$ being the number of bins receiving at least one ball. This balls-and-bins random process can be modeled by choosing a random hash function $h \in \mathcal{H}_A([A],[K])$, i.e. $h$ acts fully independently on the $A$ balls, and letting $X = |i \in [K] : h^{-1}(i) \neq \emptyset|$. When analyzing our $F_0$ algorithm, we require an understanding of how $X$ behaves when $h \in \mathcal{H}_k([A],[K])$ for $k \ll A$.

Henceforth, we let $X_i$ denote the random variable indicating that at least one ball lands in bin $i$ under a truly random hash function $h$, so that $X = \sum_{i=1}^{K} X_i$.

The following fact is standard.

FACT 1. $\mathbf{E}[X] = K\left(1 - \left(1 - \frac{1}{K}\right)^A\right)$

The proof of the following lemma is deferred to the full version due to space constraints.

LEMMA 1. *If* $100 \le A \le K/20$, *then* $\mathbf{Var}[X] < 4A^2/K$.

We now state a lemma that $k$-wise independence for small $k$ suffices to preserve $\mathbf{E}[X]$ to within $1 \pm \varepsilon$, and to preserve $\mathbf{Var}[X]$ to within an additive $\varepsilon^2$. We note that item (1) in the following lemma was already shown in [4, Lemma 1] but with a stated requirement of $k = \Omega(\log(1/\varepsilon))$, though their proof actually seems to only require $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$. Our proof of item (1) also only requires this $k$, but we require dependence on $K$ in our proof of item (2). The proof of the following lemma is in Section A.1, and is via approximate inclusion-exclusion.

LEMMA 2. *There exists some constant* $\varepsilon_0$ *such that the following holds for* $\varepsilon \le \varepsilon_0$. *Let* $A$ *balls be mapped into* $K$ *bins using a random* $h \in \mathcal{H}_{2(k+1)}([A],[K])$, *where* $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$ *for a sufficiently large constant* $c > 0$. *Suppose* $1 \le A \le K$. *For* $i \in [K]$, *let* $X'_i$ *be an indicator variable which is 1 if and only if there exists at least one ball mapped to bin* $i$ *by* $h$. *Let* $X' = \sum_{i=1}^{K} X'_i$. *Then the following hold:*

*(1).* $|\mathbf{E}[X'] - \mathbf{E}[X]| \le \varepsilon\mathbf{E}[X]$

*(2).* $\mathbf{Var}[X'] - \mathbf{Var}[X] \le \varepsilon^2$

We now give a consequence of the above lemma.

LEMMA 3. *There exists a constant $\varepsilon_0$ such that the following holds. Let $X'$ be as in Lemma 2, and also assume $100 \le A \le K/20$ with $K = 1/\varepsilon^2$ and $\varepsilon \le \varepsilon_0$. Then*

$$\mathbf{Pr}[|X' - \mathbf{E}[X]| \le 8\varepsilon \mathbf{E}[X]] \ge 4/5$$

· PROOF. Observe that

$$\begin{aligned}\mathbf{E}[X] &\ge (1/\varepsilon^2)\left(1 - \left(1 - A\varepsilon^2 + \binom{A}{2}\varepsilon^4\right)\right)\\ &= (1/\varepsilon^2)\left(A\varepsilon^2 - \binom{A}{2}\varepsilon^4\right)\\ &\ge (39/40)A,\end{aligned}$$

since $A \le 1/(20\varepsilon^2)$.

By Lemma 2 we have $\mathbf{E}[X'] \ge (1-\varepsilon)\mathbf{E}[X] > (9/10)A$, and additionally using Lemma 1 we have that $\mathbf{Var}[X'] \le \mathbf{Var}[X] + \varepsilon^2 \le 5\varepsilon^2 A^2$. Set $\varepsilon' = 7\varepsilon$. Applying Chebyshev's inequality,

$$\begin{aligned}&\mathbf{Pr}[|X' - \mathbf{E}[X']| \ge (10/11)\varepsilon' \mathbf{E}[X']]\\ &\quad \le \mathbf{Var}[X']/((10/11)^2(\varepsilon')^2\mathbf{E}^2[X'])\\ &\quad \le 5 \cdot A^2\varepsilon^2/((10/11)^2(\varepsilon')^2(9/10)^2A^2)\\ &\quad < (13/2)\varepsilon^2/(10\varepsilon'/11)^2\\ &\quad < 1/5\end{aligned}$$

Thus, with probability at least 1/5, by the triangle inequality and Lemma 2 we have $|X' - \mathbf{E}[X]| \le |X' - \mathbf{E}[X']| + |\mathbf{E}[X'] - \mathbf{E}[X]| \le 8\varepsilon\mathbf{E}[X]$. □

# 3. $F_0$ ESTIMATION ALGORITHM

In this section we describe our $F_0$-estimation algorithm. Our algorithm requires, in part, a constant-factor approximation to $F_0$ at every point in the stream in which $F_0$ is sufficiently large. We describe a subroutine ROUGHESTIMATOR in Section 3.1 which provides this, using $O(\log(n))$ space, then we give our full algorithm in Section 3.2.

We remark that the algorithm we give in Section 3.2 is space-optimal, but is not described in a way that achieves $O(1)$ worst-case update and reporting times. In Section 3.4, we describe modifications to achieve optimal running times while preserving space-optimality.

We note that several previous algorithms could give a constant-factor approximation to $F_0$ with success probability 2/3 using $O(\log(n))$ space. To understand why our guarantees from ROUGHESTIMATOR are different, one should pay particular attention to the quantifiers. In previous algorithms, it was guaranteed that there exists a constant $c > 0$ such that at any *particular* point $t$ in the stream, with probability at least $1 - \delta$ the output $\widetilde{F}_0(t)$ is in $[F_0(t), cF_0(t)]$, with the space used being $O(\log(n)\log(1/\delta))$. To then guarantee $\widetilde{F}_0(t) \in [F_0(t), cF_0(t)]$ *for all* $t \in [m]$ with probability 2/3, one should set $\delta = 1/(3m)$ to then union bound over all $t$, giving an overall space bound of $O(\log(n)\log(m))$. Meanwhile, in our subroutine ROUGHESTIMATOR, we ensure that with probability 2/3, $\widetilde{F}_0(t) \in [F_0(t), cF_0(t)]$ for all $t \in [m]$ simultaneously, and the overall space used is $O(\log(n))$.

## 3.1 RoughEstimator

We now show that ROUGHESTIMATOR (Figure 2) with probability $1 - o(1)$ (as $n \to \infty$) outputs a constant-factor approximation to $F_0(t)$ for every $t$ in which $F_0(t)$ is sufficiently large. That is, if our estimate of $F_0(t)$ is $\widetilde{F}_0(t)$,

$$\mathbf{Pr}[\forall t \in [m] \text{ s.t. } F_0 \ge K_{\mathrm{RE}},\ \widetilde{F}_0(t) = \Theta(F_0(t))] = 1 - o(1),$$

where $K_{\mathrm{RE}}$ is as in Figure 2.

THEOREM 1. *With probability $1 - o(1)$, the output $\widetilde{F}_0$ of* ROUGHESTIMATOR *satisfies $F_0(t) \le \widetilde{F}_0(t) \le 8F_0(t)$ for every $t \in [m]$ with $F_0(t) \ge K_{\mathrm{RE}}$ simultaneously. The space used is $O(\log(n))$.*

PROOF. We first analyze space. The counters in total take $O(K_{\mathrm{RE}} \log\log(n)) = O(\log(n))$ bits. The hash functions $h_1^j, h_2^j$ each take $O(\log(n))$ bits. The hash functions $h_3^j$ take $O(K_{\mathrm{RE}} \log(K_{\mathrm{RE}})) = O(\log(n))$ bits.

We now analyze correctness.

LEMMA 4. *For any fixed point $t$ in the stream with $F_0(t) \ge K_{\mathrm{RE}}$, and fixed $j \in [3]$, with probability $1 - O(1/K_{\mathrm{RE}})$ we have $F_0(t) \le \widetilde{F}_0^j(t) \le 4F_0(t)$.*

PROOF. The algorithm ROUGHESTIMATOR of Figure 2 can be seen as taking the median output of three instantiations of a subroutine, where each subroutine has $K_{\mathrm{RE}}$ counters $C_1, \ldots, C_{K_{\mathrm{RE}}}$, hash functions $h_1, h_2, h_3$, and defined quantities $T_r(t) = |\{i : C_i(t) \ge r\}|$, where $C_i(t)$ is the state of counter $C_i$ at time $t$. We show that this subroutine outputs a value $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$ with probability $1 - O(1/K_{\mathrm{RE}})$.

Define $I_r(t) \subseteq I(t)$ as the set of $i \in I(t)$ with $\mathrm{lsb}(h_1(i)) \ge r$. Note $|I_r(t)|$ is a random variable, and

$$\mathbf{E}[|I_r(t)|] = \frac{F_0(t)}{2^r},\ \mathbf{Var}[|I_r(t)|] = \frac{F_0(t)}{2^r} - \frac{F_0(t)}{2^{2r}} \le \mathbf{E}[|I_r(t)|],$$

with the latter using 2-wise independence of $h_1$. Then by Chebyshev's inequality,

$$\mathbf{Pr}\left[\left||I_r(t)| - \frac{F_0(t)}{2^r}\right| \ge q \cdot \frac{F_0(t)}{2^r}\right] \le \frac{1}{q^2 \cdot \mathbf{E}[|I_r(t)|]}. \quad (1)$$

Since $F_0(t) \ge K_{\mathrm{RE}}$, there exists an $r' \in [0, \log n]$ such that $K_{\mathrm{RE}}/2 \le \mathbf{E}[|I_{r'}(t)|] < K_{\mathrm{RE}}$. We condition on the event $\mathcal{E}$ that $K_{\mathrm{RE}}/3 \le I_{r'}(t) \le 4K_{\mathrm{RE}}/3$, and note

$$\mathbf{Pr}[\mathcal{E}] = 1 - O(1/K_{\mathrm{RE}})$$

by Eq. (1). We also condition on the event $\mathcal{E}'$ that for all $r'' > r' + 1$, $I_{r''}(t) \le 7K_{\mathrm{RE}}/24$. Applying Eq. (1) with $r = r' + 2$ and using that $I_{r+1}(t) \subseteq I_r(t)$,

$$\mathbf{Pr}[\mathcal{E}'] \ge 1 - O(1/K_{\mathrm{RE}}).$$

We now define two more events. The first is the event $\mathcal{E}''$ that $T_{r'}(t) \ge \rho K_{\mathrm{RE}}$. The second is the event $\mathcal{E}'''$ that $T_{r''}(t) < \rho K_{\mathrm{RE}}$ for all $r'' > r' + 1$. Note that if $\mathcal{E}'' \wedge \mathcal{E}'''$ holds, then $F_0(t) \le \widetilde{F}_0(t) \le 4F_0(t)$. We now show that these events hold with large probability.

Define the event $\mathcal{A}$ that the indices in $I_{r'}(t)$ are perfectly hashed under $h_2$, and the event $\mathcal{A}'$ that the indices in $I_{r'+2}(t)$ are perfectly hashed under $h_2$. Then

$$\mathbf{Pr}[\mathcal{A} \mid \mathcal{E}] \ge 1 - O(1/K_{\mathrm{RE}}).$$

and similarly for $\mathbf{Pr}[\mathcal{A}' \mid \mathcal{E}']$.

Note that, conditioned on $\mathcal{E} \wedge \mathcal{A}$, $T_{r'}(t)$ is distributed exactly as the number of non-empty bins when throwing $|I_{r'}(t)|$ balls uniformly at random into $K_{\mathrm{RE}}$ bins. This is

1. Set $K_{\mathrm{RE}} = \max\{8, \log(n)/\log\log(n)\}$.
2. Initialize $3K_{\mathrm{RE}}$ counters $C_1^j, \ldots, C_{K_{\mathrm{RE}}}^j$ to $-1$ for $j \in [3]$.
3. Pick random $h_1^j \in \mathcal{H}_2([n], [0, n-1])$, $h_2^j \in \mathcal{H}_2([n], [{K_{\mathrm{RE}}}^3])$, $h_3^j \in \mathcal{H}_{2K_{\mathrm{RE}}}([{K_{\mathrm{RE}}}^3], [K_{\mathrm{RE}}])$ for $j \in [3]$.
4. **Update(i):** For each $j \in [3]$, set $C^j_{h_3^j(h_2^j(i))} \leftarrow \max\left\{C^j_{h_3^j(h_2^j(i))}, \mathrm{lsb}(h_1^j(i))\right\}$.
5. **Estimator:** For integer $r \geq 0$, define $T_r^j = |\{i : C_i^j \geq r\}|$.
   For the largest $r = r^*$ with $T_r^j \geq \rho K_{\mathrm{RE}}$, set $\widetilde{F}_0^j = 2^{r^*} K_{\mathrm{RE}}$. If no such $r$ exists, $\widetilde{F}_0^j = -1$.
   Output $\widetilde{F}_0 = \mathrm{median}\{\widetilde{F}_0^1, \widetilde{F}_0^2, \widetilde{F}_0^3\}$.

**Figure 2: RoughEstimator pseudocode. With probability $1 - o(1)$, $\widetilde{F}_0(t) = \Theta(F_0(t))$ at every point $t$ in the stream for which $F_0(t) \geq K_{\mathrm{RE}}$. The value $\rho$ is $.99 \cdot (1 - e^{-1/3})$.**

because, conditioned on $\mathcal{E} \wedge \mathcal{A}$, there are no collisions of members of $I_{r'}(t)$ under $h_2$, and the independence of $h_3$ is larger than $|I_{r'}(t)|$. Thus,

$$\mathbf{E}[T_{r'}(t) \mid \mathcal{E} \wedge \mathcal{A}] = \left(1 - \left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{|I_{r'}(t)|}\right) K_{\mathrm{RE}}.$$

The same argument applies for the conditional expectation $\mathbf{E}[T_{r''}(t) \mid \mathcal{E}' \wedge \mathcal{A}']$ for $r'' > r' + 1$. Call these conditional expectations $E_r$. Then since $(1-1/n)/e \leq (1-1/n)^n \leq 1/e$ for all real $n \geq 1$ (see Proposition B.3 of [28]), we have that $E_r/K_{\mathrm{RE}}$ lies in the interval

$$\left[\left(1 - e^{-\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\right), \left(1 - e^{-\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\right)\right]$$

Thus for $r'' > r' + 1$,

$$E_{r''} \leq \left(1 - e^{-7/24}\left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{7/24}\right) K_{\mathrm{RE}}, \text{ and}$$

$$E_{r'} \geq \left(1 - e^{-1/3}\right) K_{\mathrm{RE}}.$$

A calculation shows that $E_{r''} < .99 E_{r'}$ since $K_{\mathrm{RE}} \geq 8$.

By negative dependence in the balls and bins random process (see [15]), the Chernoff bound applies to $T_r(t)$ and thus

$$\mathbf{Pr}\left[|T_{r'}(t) - E_{r'}| \geq \epsilon E_{r'} \mid \mathcal{E} \wedge \mathcal{A}\right] \leq 2e^{-\epsilon^2 E_{r'}/3}$$

for any $\epsilon > 0$, and thus by taking $\epsilon$ a small enough constant,

$$\mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E} \wedge \mathcal{A}] \geq 1 - e^{-\Omega(K_{\mathrm{RE}})}.$$

We also have, for $r'' > r' + 1$,

$$\mathbf{Pr}[\mathcal{E}''' \mid \mathcal{E}' \wedge \mathcal{A}'] = \mathbf{Pr}\left[T_{r''}(t) \geq \rho K_{\mathrm{RE}} \mid \mathcal{E}' \wedge \mathcal{A}'\right] \geq 1 - e^{-\Omega(K_{\mathrm{RE}})}.$$

Thus, overall,

$$\begin{aligned}
\mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E}'''] &\geq \mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E}''' \wedge \mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{A} \wedge \mathcal{A}'] \\
&\geq \mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E} \wedge \mathcal{A}] + \mathbf{Pr}[\mathcal{E}''' \wedge \mathcal{E}' \wedge \mathcal{A}'] - 1 \\
&= \mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E} \wedge \mathcal{A}] \cdot \mathbf{Pr}[\mathcal{A} \mid \mathcal{E}] \cdot \mathbf{Pr}[\mathcal{E}] \\
&\quad + \mathbf{Pr}[\mathcal{E}''' \mid \mathcal{E}' \wedge \mathcal{A}'] \cdot \mathbf{Pr}[\mathcal{A}' \mid \mathcal{E}'] \cdot \mathbf{Pr}[\mathcal{E}'] \\
&\quad - 1 \\
&\geq 1 - O(1/K_{\mathrm{RE}})
\end{aligned}$$

□

Now, note that for any $t \in [m]$, if $\widetilde{F}_0^j(t)$ is a 4-approximation to $F_0(t)$ for at least two values of $j$, then $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$. Thus by Lemma 4, $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$ with probability $1 - O(1/{K_{\mathrm{RE}}}^2)$. Let $t_r$ be the first time in the stream when $F_0(t_r) = 2^r$ (if no such time exists, let $t_r = \infty$). Then by a union bound, our estimate of $\widetilde{F}_0(t_r)$ is in $[F_0(t_r), 4F_0(t_r)]$ at all $t_r$ for $r \in [0, \log n]$ with probability $1 - O(\log(n)/{K_{\mathrm{RE}}}^2) = 1 - o(1)$. Now, observe that our estimates $\widetilde{F}_0(t)$ can only ever increase with $t$. Thus, if our estimate is in $[F_0(t_r), 4F_0(t_r)]$ at all points $t_r$, then it is in $[F_0(t), 8F_0(t)]$ for all $t \in [m]$. This concludes our proof.

## 3.2 Full algorithm

In this section we analyze our main algorithm (Figure 3), which $(1 \pm O(\varepsilon))$-approximates $F_0$ with 11/20 probability. We again point out that the implementation described in Figure 3 is not our final algorithm which achieves $O(1)$ update and reporting times; the final optimal algorithm is a modification of Figure 3, described in Section 3.4. We assume throughout that $F_0 \geq K/32$ and deal with the case of small $F_0$ in Section 3.3. The space used is $O(\varepsilon^{-2} + \log(n))$ bits. Note that the 5/8 can be boosted to $1 - \delta$ for arbitrary $\delta > 0$ by running $O(\log(1/\delta))$ instantiations of our algorithm in parallel and returning the median estimate of $F_0$. Also, the $O(\varepsilon)$ term in the error guarantee can be made $\varepsilon$ by running the algorithm with $\varepsilon' = \varepsilon/C$ for a sufficiently large constant $C$. Throughout this section we without loss of generality assume $n$ is larger than some constant $n_0$, and $1/\varepsilon^2 \geq C\log(n)$ for a constant $C$ of our choice, and is a power of 2. If one desires a $(1 \pm \varepsilon)$-approximation for $\varepsilon > 1/\sqrt{C\log(n)}$, we simply run our algorithm with $\varepsilon = 1/\sqrt{C\log(n)}$, which worsens our promised space bound by at most a constant factor.

The algorithm of Figure 3 works as follows. We maintain $K = 1/\varepsilon^2$ counters $C_1, \ldots, C_K$ as well as three values $A, b, \mathrm{est}$. Each index is hashed to some level between 0 and $\log(n)$, based on the least significant bit of its hashed value, and is also hashed to one of the counters. Each counter maintains the deepest level of an item that was hashed to it. Up until this point, this information being kept is identical as in the LogLog [16] and HyperLogLog [19] algorithms (though our analysis will not require that the hash functions be truly random). The value $A$ keeps track of the amount of storage required to store all the $C_i$, and our algorithm fails if this value ever becomes much larger than a constant times $K$ (which we show does not happen with large probability).

1. Set $K = 1/\varepsilon^2$.
2. Initialize $K$ counters $C_1, \ldots, C_K$ to $-1$.
3. Pick random $h_1 \in \mathcal{H}_2([n],[0,n-1])$, $h_2 \in \mathcal{H}_2([n],[K^3])$, $h_3 \in \mathcal{H}_k([K^3],[K])$ for $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$.
4. Initialize $A, b, \mathrm{est} = 0$.
5. Run an instantiation RE of RoughEstimator.
6. **Update(i):** Set $x \leftarrow \max\{C_{h_3(h_2(i))}, \mathrm{lsb}(h_1(i)) - b\}$.
   Set $A \leftarrow A - \lceil \log(2 + C_{h_3(h_2(i))}) \rceil + \lceil \log(2+x) \rceil$.
   If $A > 3K$, Output FAIL.
   Set $C_{h_3(h_2(i))} \leftarrow x$. Also feed $i$ to RE.
   Let $R$ be the output of RE.

   **if** $R > 2^{\mathrm{est}}$:

   (a) $\mathrm{est} \leftarrow \log(R)$, $b_{\mathrm{new}} \leftarrow \max\{0, \mathrm{est} - \log(K/32)\}$.

   (b) For each $j \in [K]$, set $C_j \leftarrow \max\{-1, C_j + b - b_{\mathrm{new}}\}$

   (c) $b \leftarrow b_{\mathrm{new}}$, $A \leftarrow \sum_{j=1}^{K} \lceil \log(C_j + 2) \rceil$.
7. **Estimator:** Define $T = |\{j : C_j \geq 0\}|$. Output $\widetilde{F}_0 = 2^b \cdot \frac{\ln(1-\frac{T}{K})}{\ln(1-\frac{1}{K})}$.

**Figure 3: $F_0$ algorithm pseudocode. With probability $11/20$, $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$.**

The value est is such that $2^{\mathrm{est}}$ is a $\Theta(1)$-approximation to $F_0$, and is obtained via RoughEstimator, and $b$ is such that we expect $F_0(t)/2^b$ to be $\Theta(K)$ at all points $t$ in the stream. Each $C_i$ then actually holds the *offset* (from $b$) of the deepest level of an item that was hashed to it; if no item of level $b$ or deeper hashed to $C_i$, then $C_i$ stores $-1$. Furthermore, the counters are bitpacked so that $C_i$ only requires $O(1 + \log(C_i))$ bits of storage (Section 3.4 states a known data structure which allows the bitpacked $C_i$ to be stored in a way that supports efficient reads and writes).

Theorem 2. *The algorithm of Figure 3 uses $O(\varepsilon^{-2} + \log(n))$ space.*

Proof. The hash functions $h_1, h_2$ each require $O(\log(n))$ bits to store. The hash function $h_3$ takes $O(k \log(K)) = O(\log^2(1/\varepsilon))$ bits. The value $b$ takes $O(\log\log n)$ bits. The value $A$ never exceeds the total number of bits to store all counters, which is $O(\varepsilon^{-2}\log(n))$, and thus $A$ can be represented in $O(\log(1/\varepsilon) + \log\log(n))$ bits. The counters $C_j$ never in total consume more than $O(1/\varepsilon^2)$ bits by construction, since we output FAIL if they ever would. □

Theorem 3. *The algorithm of Figure 3 outputs a value which is $(1 \pm O(\varepsilon))F_0$ with probability at least $11/20$ as long as $F_0 \geq K/32$.*

Proof. Let $\widetilde{F}_0^{\mathrm{RE}}(t)$ be the estimate of $F_0$ offered by RE at time $t$. Throughout this proof we condition on the event $\mathcal{E}$ that $F_0(t) \leq \widetilde{F}_0^{\mathrm{RE}}(t) \leq 8F_0(t)$ for all $t \in [m]$, which occurs with probability $1 - o(1)$ by Theorem 1.

We first show that the algorithm does not output FAIL with large probability. Note $A$ is always $\sum_{i=1}^{K} \lceil \log(C_i + 2) \rceil$, and we must thus show that with large probability this quantity is at most $3K$ at all points in the stream. Let $A(t)$ be the value of $A$ at time $t$ (before running steps (a)-(c)), and similarly define $C_j(t)$. We condition on the randomness used by RE, which is independent from the remaining parts of the algorithm. Let $t_1, \ldots, t_{r-1}$ be the points in the stream where the output of RE changes, i.e $\widetilde{F}_0^{\mathrm{RE}}(t_j - 1) \neq \widetilde{F}_0^{\mathrm{RE}}(t_j)$ for all $j \in [r-1]$, and define $t_r = m$. We note that $A(t)$ takes on its maximum value for $t = t_j$ for some $j \in [r]$, and thus it suffices to show that $A(t_j) \leq 3K$ for all $j \in [r]$. We furthermore note that $r \leq \log(n) + 3$ since $\widetilde{F}_0^{\mathrm{RE}}(t)$ is weakly increasing, only increases in powers of 2, and is always between 1 and $8F_0 \leq 8n$ given that $\mathcal{E}$ occurs. Now,

$$\begin{aligned} A(t) &\leq K + \sum_{i=1}^{K} \log(C_i(t) + 2) \\ &\leq K + K \cdot \log\left(\frac{\sum_{i=1}^{K} C_i(t)}{K} + 2\right) \end{aligned}$$

with the last inequality using concavity of the logarithm and Jensen's inequality. It thus suffices to show that, with large probability, $\sum_{i=1}^{K} C_i(t_j) \leq 2K$ for all $j \in [r]$.

Fix some $t = t_j$ for $j \in [r]$. For $i \in I(t)$, let $X_i(t)$ be the random variable $\max\{-1, \mathrm{lsb}(h_1(i)) - b\}$, and let $X(t) = \sum_{i \in I(t)} X_i(t)$. Note $\sum_{i=1}^{K} C_i(t) \leq X(t)$, and thus it suffices to lower bound $\mathbf{Pr}[X(t) \leq 2K]$.

We have that $X_i(t)$ equals $s$ with probability $1/2^{b+s+1}$ for $0 \leq s < \log(n) - b$, equals $\log(n) - b$ with probability $1/n$, and equals $-1$ with the remaining probability mass. Thus

$$\mathbf{E}[X(t)] \leq F_0(t) \cdot \left(1/n + \sum_{s=0}^{\log(n)-b-1} 2^{-(b+s+1)}\right) = F_0(t)/2^b.$$

Furthermore, by choice of $h_1$ the $X_i(t)$ are pairwise independent, and thus $\mathbf{Var}[X(t)] \leq \mathbf{E}[X(t)]$ since $\mathbf{Var}[X_i(t)] \leq \mathbf{E}[X_i(t)]$. Then by Chebyshev's inequality,

$$\mathbf{Pr}[X(t) > 2K] < \frac{F_0(t)}{2^b \cdot (2K - F_0(t)/2^b)^2}$$

Conditioned on $\mathcal{E}$, $K/256 \leq F_0(t)/2^b \leq K/32$, implying the above probability is at most $1/(32K)$. Then by a union bound over all $t_j$ for $j \in [r]$, we have that $X(t) \leq 2K$ for all $j \in [r]$ with probability at least $1 - r/(32K) \geq 1 - 1/32$ by our assumed upper bound on $\varepsilon$, implying we output FAIL with probability at most $1/32$.

We now show that the output from Step 7 in Figure 3 is

$(1 \pm O(\varepsilon))F_0$ with probability 11/16. Let $\mathcal{A}$ be the algorithm in Figure 3, and let $\mathcal{A}'$ be the same algorithm, but without the third line in the update rule (i.e., $\mathcal{A}'$ never outputs FAIL). We first show that the output $\widetilde{F}_0$ of $\mathcal{A}'$ is $(1 \pm O(\varepsilon))F_0$ with probability 5/8. Let $I_b$ be the set of indices $i \in I$ such that $\mathrm{lsb}(i) \geq b$. Then $\mathbf{E}[|I_b|] = F_0/2^b$, and $\mathbf{Var}[|I_b|] \leq \mathbf{E}[|I_b|]$, with the last inequality in part using pairwise independence of $h_1$. We note that conditioned on $\mathcal{E}$, we have

$$K/256 \leq \mathbf{E}[|I_b|] \leq K/32$$

Let $\mathcal{E}'$ be the event that $K/300 \leq |I_b| \leq K/20$. Then by Chebyshev's inequality,

$$\mathbf{Pr}[\mathcal{E}' \mid \mathcal{E}] \geq 1 - O(1/K) = 1 - o(1).$$

Also, if we let $\mathcal{E}''$ be the event that $I_b$ is perfectly hashed under $h_2$, then pairwise independence of $h_2$ gives

$$\mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E}'] \geq 1 - O(1/K) = 1 - o(1).$$

Now, conditioned on $\mathcal{E}' \wedge \mathcal{E}''$, we have that $T$ is a random variable counting the number of bins hit by at least one ball under a $k$-wise independent hash function, where there are $B = |I_b|$ balls, $K$ bins, and $k = \Omega(\log(K/\varepsilon)/\log\log(K/\varepsilon))$. Then by Lemma 3, $T = (1 \pm 8\varepsilon)(1 - (1 - 1/K)^B)K$ with 4/5 probability, in which case

$$\ln(1 - T/K) = \ln((1 - 1/K)^B \pm 8\varepsilon(1 - (1 - 1/K)^B))$$

Conditioned on $\mathcal{E}'$, $(1 - 1/K)^B = \Theta(1)$, and thus the above is $\ln((1 \pm O(\varepsilon))(1 - 1/K)^B) = B\ln(1 - 1/K) \pm O(\varepsilon)$ since $\ln(1 + x) = O(|x|)$ for $|x| < 1/2$, and thus

$$\widetilde{F}_0 = B \cdot 2^b \pm O(\varepsilon \cdot 2^b K). \qquad (2)$$

Conditioned on $\mathcal{E}$, we have that $2^b \leq 256F_0/K$, and thus the error term in Eq. (2) is $O(\varepsilon F_0)$. Also, $\mathbf{E}[B] = F_0/2^b$, which is at least $K/256$ conditioned on $\mathcal{E}$. Thus by pairwise independence of $h_1$, Chebyshev's inequality implies

$$\mathbf{Pr}[|B - \mathbf{E}[B]| \geq c/\sqrt{K}] \leq \frac{\mathbf{E}[B]}{(c^2/K) \cdot \mathbf{E}[B]^2} \leq \left(\frac{16}{c}\right)^2$$

since $\mathbf{Var}[B] \leq \mathbf{E}[B]$, which we can make an arbitrarily small constant by setting $c$ to be a large constant. Note that $1/\sqrt{K}$ is just $\varepsilon$, and thus we have that $B = (1 \pm O(\varepsilon))F_0/2^b$ with arbitrarily large constant probability.

Putting everything together, we have that, conditioned on $\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''$, $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$ with probability at least $4/5 - \delta$ for any constant $\delta > 0$ of our choice, e.g. $\delta = 1/5$. Since $\mathbf{Pr}[\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''] \geq 1 - o(1)$, we thus have

$$\mathbf{Pr}[\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0] \geq 3/5 - o(1).$$

Note our algorithm in Figure 3 succeeds as long as (1) we do not output FAIL, and (2) $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$, and thus overall we succeed with probability at least $1 - \frac{2}{5} - o(1) - \frac{1}{32} > \frac{11}{20}$. □

## 3.3 Handling small $F_0$

In Section 3.2, we assumed that $F_0 = \Omega(K)$ for $K = 1/\varepsilon^2$ (specifically, $F_0 \geq K/32$). In this subsection, we show how to deal with the case that $F_0$ is small, by running a similar (but simpler) algorithm to that of Figure 3 in parallel.

The case $F_0 < 100$ can be dealt with simply by keeping the first 100 distinct indices seen in the stream in memory, taking $O(\log(n))$ space.

For the case $F_0 \geq 100$ we can apply Lemma 3 as was done in the proof of Theorem 3. We maintain $K' = 2K$ bits $B_1, \ldots, B_{K'}$ in parallel, initialized to 0. When seeing an index $i$ in the stream, in addition to carrying out Step 6 of Figure 3, we also set $B_{h_3(h_2(i))}$ to 1 ($h_3$ can be taken to have range $K' = 2K$, and its evaluation can be taken modulo $K$ when used in Figure 3 to have a size-$K$ range). Let $t_0$ be the smallest $t \in [m]$ with $F_0(t) = K'/64$, and $t_1$ be the smallest $t \in [m]$ with $F_0(t) = K'/32$ (if no such $t_i$ exist, set them to $\infty$). Define $T_B(t) = |\{i : B_i(t) = 1\}|$, and define $\widetilde{F}_0^B(t) = \ln(1 - T_B(t)/K')/\ln(1 - 1/K')$. Then by similar calculations as in Theorem 3 and a union bound over $t_0, t_1$, $\mathbf{Pr}[\widetilde{F}_0^B(t_i) = (1 \pm O(\varepsilon))F_0(t_i)$ for $i \in \{0, 1\}]$ is at least $1 - 2 \cdot (1/5) - o(1) = 3/5 - o(1)$. Noting that $\widetilde{F}_0^B(t)$ monotonically increases with $t$, we can do the following: for $t$ with $\widetilde{F}_0^B(t) \geq K'/32 = K/16$, we output the estimator from Figure 3; else, we output $\widetilde{F}_0^B(t)$. We summarize this section with the following theorem.

THEOREM 4. *Let $\delta > 0$ be any fixed constant, and $\varepsilon > 0$ be given. There is a subroutine requiring $O(\varepsilon^{-2} + \log(n))$ space which with probability $1 - \delta$ satisfies the property that there is some $t' \in [m]$ satisfying: (1) for any fixed $t < t'$, $(1 \pm O(\varepsilon))F_0$ is output, and (2) for any $t \geq t'$ the subroutine outputs* LARGE*, and we are guaranteed $F_0(t) \geq 1/(16\varepsilon^2)$.'*

## 3.4 Running time

In this subsection we discuss an implementation of our $F_0$ algorithm in Figure 3 with $O(1)$ update and reporting times. We first state a few theorems from previous works.

THEOREM 5 (BRODNIK [8], FREDMAN AND WILLARD [21]). *The least and most significant bits of an integer fitting in a machine word can be computed in constant time.*

The next two theorems give hash families which have strong independence properties while only requiring $O(1)$ evaluation time (recall that the $k$-wise independent hash functions of Carter and Wegman require $\Theta(k)$ evaluation time).

THEOREM 6 (PAGH AND PAGH [31, THEOREM 1.1]). *Let $S \subseteq U = [u]$ be a set of $z > 1$ elements, and let $V = [v]$, with $1 < v \leq u$. Suppose the machine word size is $\Omega(\log(u))$. For any constant $c > 0$ there is word RAM algorithm that, using time $\log(z)\log^{O(1)}(v)$ and $O(\log(z) + \log\log(u))$ bits of space, selects a family $\mathcal{H}$ of functions from $U$ to $V$ (independent of $S$) such that:*

1. *With probability $1 - O(1/z^c)$, $\mathcal{H}$ is $z$-wise independent when restricted to $S$.*

2. *Any $h \in \mathcal{H}$ can be represented by a RAM data structure using $O(z\log(v))$ bits of space, and $h$ can be evaluated in constant time after an initialization step taking $O(z)$ time.*

The following is a corollary of Theorem 2.16 in [35].

THEOREM 7 (SIEGEL [35]). *Let $U = [u]$ and $V = [v]$ with $u = v^c$ for some constant $c \geq 1$, where the machine word size is $\Omega(\log(v))$. Suppose one wants a $k(v)$-wise independent hash family $\mathcal{H}$ of functions mapping $U$ to $V$ for $k(v) = v^{o(1)}$. For any constant $\epsilon > 0$ there is a randomized procedure for constructing such an $\mathcal{H}$ which succeeds* *with probability $1 - 1/v^{\epsilon}$, taking $v^{\epsilon}$ bits of space. A random $h \in \mathcal{H}$ can be selected using $v^{\epsilon}$ bits of random seed, and $h$ can be evaluated in $O(1)$ time.*

We now describe a fast version of RoughEstimator.

Lemma 5. RoughEstimator *can be implemented with $O(1)$ worst-case update and reporting times, at the expense of only giving a 16-approximation to $F_0(t)$ for every $t \in [m]$ with $F_0(t) \geq K_{\mathrm{RE}}$, for $K_{\mathrm{RE}}$ as in Figure 2.*

Proof. We first discuss update time. We replace each $h_3^j$ with a random function from the hash family $\mathcal{H}$ of Theorem 6 with $z = 2K_{\mathrm{RE}}$, $u = {K_{\mathrm{RE}}}^3$, $v = K_{\mathrm{RE}}$. The constant $c$ in Item 1 of Theorem 6 is chosen to be 1, so that each $h_3^j$ is uniform on any given subset of $z$ items of $[u]$ with probability $1 - O(1/K_{\mathrm{RE}})$. Note that the proof of correctness of RoughEstimator (Theorem 1) only relied on the $h_3^j$ being uniform on some unknown set of $4K_{\mathrm{RE}}/3 < 2K_{\mathrm{RE}}$ indices with probability $1 - O(1/K_{\mathrm{RE}})$ (namely, those indices in $I_{r'}(t)$). The space required to store any $h \in \mathcal{H}$ is $z \log(v) = O(\log(n))$, which does not increase our space bound for RoughEstimator. Updates then require computing a least significant bit, and computing the $h_1^j, h_2^j, h_3^j$, all taking constant time.

For reporting time, in addition to the information maintained in Figure 2, we also maintain three sets of counters $A_0^j, A_1^j, A_2^j, A_3^j, A_4^j$ for $j \in [3]$. For a fixed $j$, the $A_i^j$ store $T_{r+i}^j$ for an $r$ we now specify. Roughly speaking, for the values of $t$ where $F_0(t) \geq K_{\mathrm{RE}}$, $r$ will be such that, conditioned on RoughEstimator working correctly, $2^r$ will always be in $[F_0(t)/2, 8F_0(t)]$. We then alter the estimator of RoughEstimator to being $2^{r+1}$.

Note that, due to Theorem 4, the output of RoughEstimator does not figure into our final $F_0$-estimator until $F_0(t) \geq (1 - O(\varepsilon))/(32\varepsilon^2)$, and thus the output of the algorithm is irrelevant before this time. We start off with $r = \log(1/(32\varepsilon^2))$. Note that $A_0^j, A_1^j, A_2^j, A_3^j, A_4^j$ can be maintained in constant time during updates. At some point $t_1$, the estimator from Section 3.3 will declare that $F_0(t_1) = (1 \pm O(\varepsilon))/(32\varepsilon^2)$, at which point we are assured $F_0(t_1) \geq 1/(64\varepsilon^2) \geq \log(n)$ (assuming $\varepsilon$ is smaller than some constant, and assuming that $1/\varepsilon^2 \geq 64\log(n)$). Similarly, we also have $F_0(t_1) \leq 1/(16\varepsilon^2) \leq 4\log(n)$. Thus, by our choice of $r$ and conditioned on the event that RoughEstimator of Figure 2 succeeds (i.e., outputs a value in $[F_0(t), 8F_0(t)]$ for all $t$ with $F_0(t) \geq K_{\mathrm{RE}}$), we can determine the median across the $j$ of the largest $r^*$ such that $T_r^j \geq \rho K_{\mathrm{RE}}$ from the $A_i^j$ and set $r(t_1) = r^*$ so that $2^{r(t_1)}$ is in $[F_0(t_1), 8F_0(t_1)]$.

Our argument henceforth is inductive: conditioned on the output of RoughEstimator from Figure 2 being correct (always in $[F_0(t), 8F_0(t)]$), $2^{r(t)}$ will always be in $[F_0(t)/2, 8F_0(t)]$ for all $t \geq t_1$, which we just saw is true for $t = t_1$. Note that conditioned on RoughEstimator being correct, its estimate of $F_0$ cannot jump by a factor more than 8 at any given point in the stream. Furthermore, if this happens, we will detect it since we store up to $A_4^j$. Thus, whenever we find that the estimate from RoughEstimator changed (say from $2^{r'}$ to $2^{r''}$), we increment $r$ by $r'' - r'$ and set each $A_i^j$ to $A_{i+r''-r'}^j$ for $i \leq 4 + r' - r''$ For $4 + r' - r'' < i \leq 4$, we recompute $A_i^j$ from scratch, by looping over the $K_{\mathrm{RE}}$ counters $C_i$. This requires $O(K_{\mathrm{RE}})$ work, but note that since $t \geq t_1$, there must be at least $K_{\mathrm{RE}}$ updates before $F_0(t)$ doubles, and thus we can afford to do $O(1)$ work toward this looping per update. In the meantime $2^r$ cannot fall below $F_0/2$. □

We will use the following "variable-bit-length array" data structure to implement the array $C$ of counters in Figure 3, which has entries whose binary representations may have unequal lengths. Specifically, in Figure 3, the bit representation of $C_i$ requires $O(1 + \log(C_i + 2))$ bits.

Definition 1 (Blandford, Blelloch [7]). *A* variable-bit-length array (VLA) *is a data structure implementing an array $C_1, \ldots, C_n$ supporting the following operations: (1)* **update**$(i, x)$ *sets the value of $C_i$ to $x$, and (2)* **read**$(i)$ *returns $C_i$. Unlike in standard arrays, the $C_i$ are allowed to have bit-representations of varying lengths, and we use* $\mathrm{len}(C_i)$ *to represent the length of the bit-representation of $C_i$.*

Theorem 8 (Blandford and Blelloch [7]). *There is a VLA data structure using $O(n + \sum_i \mathrm{len}(C_i))$ space to store $n$ elements, supporting worst-case $O(1)$ updates and reads, under the assumptions that (1) $\mathrm{len}(C_i) \leq w$ for all $i$, and (2) $w \geq \log(\mathcal{M})$. Here $w$ is the machine word size, and $\mathcal{M}$ is the amount of memory available to the VLA.*

We now give a time-optimal version of Figure 3.

Theorem 9. *The algorithm of Figure 3 can be implemented with $O(1)$ worst-case update and reporting times.*

Proof. For update time, we select $h_3$ from the hash family of Theorem 7, which requires $O(1/\varepsilon^{\epsilon})$ space for arbitrarily small $\epsilon > 0$ of our choosing (say, $\epsilon = 1$), and thus this space is dominated by other parts of the algorithm. We then can evaluate $h_1, h_2, h_3$ in constant time, as well as compute the required least significant bit in constant time. Updating $A$ requires computing the ceiling of a base-2 logarithm, but this is just a most significant bit computation which we can do in $O(1)$ time. We can also read and write the $C_j$ in constant time whilst using the same asymptotic space by Theorem 8.

What remains is to handle the **if** statement for when $R > 2^{\mathrm{est}}$. Note that a naïve implementation would require $O(K)$ time. Though this **if** statement occurs infrequently enough that one could show $O(1)$ amortized update time, we instead show the stronger statement that an implementation is possible with $O(1)$ worst case update time. The idea is similar to that in the proof of Lemma 5: when $b_{\mathrm{new}}$ changes, it cannot change more than a constant number of times again in the next $O(K)$ updates, and so we can spread the $O(K)$ required work over the next $O(K)$ stream updates, doing a constant amount of work each update.

Specifically, note that $b_{\mathrm{new}}$ only ever changes for times $t$ when $R(t) > 2^{\mathrm{est(t)}} \geq K/16$, conditioned on the subroutine of Theorem 4 succeeding, implying that $F_0(t) \geq K/256$, and thus there must be at least $K/256$ updates for $F_0(t)$ to double. Since RoughEstimator always provides an 8-approximation, est can only increase by at most 3 in the next $K/256$ stream updates. We will maintain a primary and secondary instantiation of our algorithm, and only the primary receives updates. Then in cases where $R > 2^{\mathrm{est}}$ and $b_{\mathrm{new}}$ changes from $b$, we copy a sufficiently large constant number of the $C_j$ (specifically, $3 \cdot 256$) for each of the next $K/256$ updates, from the primary to secondary structure, performing the update $C_j \leftarrow \max\{-1, C_j + b - b_{\mathrm{new}}\}$ in the secondary structure. If RoughEstimator fails and est changes by more than 3 in the next $K/256$ updates, we output FAIL. Meanwhile, during this copy phase, we process new stream updates in both the primary and secondary structures, and we answer updates from the primary structure. The analysis of correctness remains virtually unchanged, since the value $2^b$ corresponding to the primary structure still remains a constant-factor approximation to $F_0$ during this copy phase.

For reporting time, note we can maintain $T = |\{i : C_i \geq 0\}|$ during updates, and thus the reporting time is the time to compute a natural logarithm, which can be made $O(1)$ via a small lookup table (see Section A.2). $\square$

# 4. $L_0$ ESTIMATION ALGORITHM

Here we give an algorithm for estimating $L_0$, the Hamming norm of a vector updated in a stream.

Our $L_0$ algorithm is based on the approach to $F_0$ estimation in Figure 4. In this approach, we maintain a $\lg(n) \times K$ bit-matrix $A$, and upon receiving an update $i$, we subsample $i$ to the row determined by the lsb of a hash evaluation, then evaluate another hash function to tell us a column and set the corresponding bit of $A$ to 1. Note that our algorithm from Section 3 is just a space-optimized implementation of this approach. Specifically, in Figure 3 we obtained a $c$-approximation $R$ to $F_0$ via RoughEstimator for $c = 8$. The value $b$ we maintained was just $\max\{0, \lg(32R/K)\}$. Then rather than explicitly maintaining $A$, we instead maintained counters $C_j$ which allowed us to deduce whether $A_{b,j} = 1$ (specifically, $A_{b,j} = 1$ iff $C_j = 0$).

The proof of correctness of the approach in Figure 4 is thus essentially identical to that of Theorem 3 (in fact simpler, since we do not have to upper bound the case of outputting FAIL), so we do not repeat it here. Thus, we need only show that the approach in Figure 4 can be implemented for some constant $c \geq 1$ in the context of $L_0$-estimation. Specifically, we must show that (a) the bit-matrix $A$ can be maintained (with large probability), and (b) we can implement the oracle in Step 4 of Figure 4 to give a $c$-approximation to $L_0$ for some constant $c \geq 1$.

We first show (a), that we can maintain the bit-matrix $A$ with large probability. In fact, note our estimate of $L_0$ only depends on one particular row $i^* = \log(16R/K)$ of $A$, so we need only ensure that we maintain row $i^*$ with large constant probability. We first give two facts.

Fact 2. *Let $t, r > 0$ be integers. Pick $h \in \mathcal{H}_2([r], [t])$. For any $S \subset [r]$ , $\mathbf{E}[\sum_{i=1}^{s} \binom{|h^{-1}(i) \cap S|}{2}] \leq |S|^2/(2t)$.*

Proof. Write $|S| = s$. Let $X_{i,j}$ indicate $h(i) = j$. By linearity of expectation, the desired expectation is then

$$t \sum_{i<i'} \mathbf{E}[X_{i,1}]\mathbf{E}[X_{i',1}] = t\binom{s}{2}\frac{1}{t^2} \leq \frac{s^2}{2t}.$$

$\square$

Fact 3. *Let $\mathbb{F}_q$ be a finite field and $v \in \mathbb{F}_q^d$ be a non-zero vector. Then, a random $w \in \mathbb{F}_q^d$ gives $\mathbf{Pr}_w[v \cdot w = 0] = 1/q$, where $v \cdot w$ is the inner product over $\mathbb{F}_q$.*

Proof. The set of vectors orthogonal to $v$ is a linear subspace $V \subset \mathbb{F}_q^d$ of dimension $d-1$ and thus has $q^{d-1}$ points. Thus, $\mathbf{Pr}[w \in V] = 1/q$. $\square$

Lemma 6. *There is a scheme which represents each $A_{i,j}$ using $O(\log(1/\varepsilon) + \log\log(mM))$ bits such that, for $i^* = \log(16R/K)$, the $(i^*)$th row of $A$ can be recovered with probability $2/3$. Furthermore, the update time and time to recover any $A_{i,j}$ are both $O(1)$.*

Proof. We represent each $A_{i,j}$ as a counter $B_{i,j}$ of $O(\log(K) + \log\log(mM))$ bits. We interpret $A_{i,j}$ as being the bit "1" if $B_{i,j}$ is non-zero; else we intrepret $A_{i,j}$ as 0. The details are as follows. We choose a prime $p$ randomly in $[D, D^3]$ for $D = 100K\log(mM)$. Notice that for $mM$ larger than some constant, by standard results on the density of primes there are at least $K^2\log^2(mM)$ primes in the interval $[D, D^3]$. Since every frequency $x_i$ is at most $mM$ in magnitude and thus has at most $\log(mM)$ prime factors, non-zero frequencies remain non-zero modulo $p$ with probability $1 - O(1/K^2)$, which we condition on occurring. We also randomly pick a vector $\mathbf{u} \in \mathbb{F}_p^K$ and $h_4 \in \mathcal{H}_2([K^3], [K])$. Upon receiving an update $(i, v)$, we increment $B_{\mathrm{lsb}(h_1(i)), h_3(h_2(i))}$ by $v \cdot \mathbf{u}_{h_4(h_2(i))}$, then reduce modulo $p$.

Define $I_{i^*} = \{i \in I : \mathrm{lsb}(i) = i^*\}$. Note that conditioned on $R \in [L_0, cL_0]$, we have $\mathbf{E}[I_{i^*}] \leq K/32$, and thus $\mathbf{Pr}[|I_{i^*}| \leq K/20] = 1 - O(1/K) = 1 - o(1)$ by Chebyshev's inequality. We condition on this event occurring. Also, since the range of $h_2$ is of size $K^3$, the indices in $I_{i^*}$ are perfectly hashed with probability $1 - O(1/K) = 1 - o(1)$, which we also condition on occurring.

Let $\mathcal{Q}$ be the event that $p$ does not divide any $|x_j|$ for $j \in I_{i^*}$. Then by a union bound, $\mathbf{Pr}[\mathcal{Q}] = 1 - O(1/K)$.

Let $\mathcal{Q}'$ be the event that $h_4(h_2(j)) \neq h_4(h_2(j'))$ for distinct $j, j' \in I_{i^*}$ with $h_3(h_2(j)) = h_3(h_2(j'))$.

Henceforth, we also condition on both $\mathcal{Q}$ and $\mathcal{Q}'$ occurring, which we later show holds with good probability. Define $J$ as the set of $j \in [K]$ such that $h_3(h_2(i)) = j$ for at least one $i \in I_{i^*}$, so that to properly represent the $A_{i^*,j}$ we should have $B_{i^*,j}$ non-zero iff $j \in J$. For each $j \in J$, $B_{i^*,j}$ can be viewed as maintaining the dot product of a non-zero vector $\mathbf{v}$, the frequency vector $x$ restricted to coordinates in $I_{i^*}$ which hashed to $j$, with a random vector $\mathbf{w}$, namely, the projection of $\mathbf{u}$ onto coordinates in $I_{i^*}$ that hashed to $j$. The vector $\mathbf{v}$ is non-zero since we condition on $\mathcal{Q}$, and $\mathbf{w}$ is random since we condition on $\mathcal{Q}'$.

Now, let $X_{i,j}$ be a random variable indicating that $h_3(h_2(j)) = h_3(h_2(j'))$ for distinct $j, j' \in I_{i^*}$. Let $X = \sum_{j<j'} X_{j,j'}$. By Fact 2 with $r = K^3$, $t = K$, and $s = |I_{i^*}| < K/20$, we have that $\mathbf{E}[X] \leq K/800$. Let $Z = \{\{j, j'\} \in \binom{I_{i^*}}{2} : h_3(h_2(j)) = h_3(h_2(j'))\}$. For $(j, j') \in Z$ let $Y_{j,j'}$ be a random variable indicating $h_4(h_2(j)) = h_4(h_2(j'))$, and let $Y = \sum_{(j,j') \in Z} Y_{j,j'}$. Then by pairwise independence of $h_4$, and the fact that we conditioned on $I_{i^*}$ being perfectly hashed under $h_2$, we have

$$\mathbf{E}[Y] = \sum_{(j,j') \in Z} \mathbf{Pr}[h_4(h_2(j)) = h_4(h_2(j'))] = |Z|/K.$$

Note $|Z| = X$. Conditioned on $X \leq 20\mathbf{E}[X] \leq K/40$, which happens with probability at least $19/20$ by Markov's inequality, we have that $\mathbf{E}[Y] \leq |Z|/K \leq 1/40$, so that $\mathbf{Pr}[Y \geq 1] \leq 1/40$. Thus, $\mathcal{Q}'$ holds with probability at least $(19/20) \cdot (39/40) > 7/8$.

Finally, by Fact 3 with $q = p$, and union bounding over all $K$ counters $B_{i^*,j}$, no $B_{i^*,j}$ for $j \in J$ is 0 with probability $1 - K/p \geq 99/100$. Thus, our scheme overall succeeds with probability $(7/8) \cdot (99/100) - o(1) > 2/3$. $\square$

We next show (b) in Section A.3, i.e. give an algorithm providing an $O(1)$-approximation to $L_0$ with $O(1)$ update and reporting times. The space used is $O(\log(n)\log\log(mM))$.

1. Set $K = 1/\varepsilon^2$.
2. Instantiate a $\lg(n) \times K$ bit-matrix $A$, initializing each $A_{i,j}$ to 0.
3. Pick random $h_1 \in \mathcal{H}_2([n],[0,n-1])$, $h_2 \in \mathcal{H}_2([n],[K^3])$, $h_3 \in \mathcal{H}_k([K^3],[K])$ for $k = \Omega(\lg(1/\varepsilon)/\lg\lg(1/\varepsilon))$.
4. Obtain a value $R \in [F_0, cF_0]$ from some oracle, for some constant $c \geq 1$.
5. **Update(i):** Set $A_{\mathrm{lsb}(h_1(i)),h_3(h_2(i))} \leftarrow 1$.
6. **Estimator:** Define $T = |\{j \in [K] : A_{\log(16R/K),j} = 1\}|$. Output $\widetilde{F}_0 = \frac{32R}{K} \cdot \frac{\ln\left(1-\frac{T}{K}\right)}{\ln\left(1-\frac{1}{K}\right)}$.

**Figure 4: An algorithm skeleton for $F_0$ estimation.**

Note that, as with our $F_0$ algorithm, we also need to have an algorithm which provides a $(1 \pm \varepsilon)$-approximation when $L_0 \ll 1/\varepsilon^2$. Just as in Section 3.3, this is done by handling the case of small $L_0$ in two cases separately: detecting and estimating when $L_0 \leq 100$, and $(1 \pm \varepsilon)$-approximating $L_0$ when $L_0 > 100$. In the former case, we can compute $L_0$ exactly with large probability by perfect hashing (see Lemma 8 in Section A.3). In the latter case, we use the same scheme as in Section 3.3, but using Lemma 6 to represent our bit array.

Putting everything together, we have the following.

Theorem 10. *There is an algorithm for $(1\pm\varepsilon)$-approximating $L_0$ using space $O(\varepsilon^{-2}\log(n)(\log(1/\varepsilon)+\log\log(mM)))$, with $2/3$ success probability, and with $O(1)$ update and reporting times.*


## Acknowledgments

We thank Nir Ailon, Erik Demaine, Piotr Indyk, T.S. Jayram, Swastik Kopparty, Rasmus Pagh, Mihai Pătraşcu, and the authors of [6] for valuable discussions and references.


## 5. REFERENCES


[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *SIGMOD*, pages 574–576, 1999.

[2] A. Akella, A. Bharambe, M. Reiter, and S. Seshan. Detecting DDoS attacks on ISP networks. In *Proc. MPDS*, 2003.

[3] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[4] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. RANDOM*, pages 1–10, 2002.

[5] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. SODA*, pages 623–632, 2002.

[6] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, pages 199–210, 2007.

[7] D. K. Blandford and G. E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Trans. Alg.*, 4(2), 2008.

[8] A. Brodnik. Computation of the least significant set bit. In *Proc. ERK*, 1993.

[9] J. Brody and A. Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *Proc. CCC*, pages 358–368, 2009.

[10] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, pages 161–180, 2005.

[11] L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[12] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[13] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.*, 15(3):529–540, 2003.

[14] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, pages 240–251, 2002.

[15] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998.

[16] M. Durand and P. Flajolet. Loglog counting of large cardinalities (extended abstract). In *Proc. ESA*, pages 605–617, 2003.

[17] C. Estan, G. Varghese, and M. E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006.

[18] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Trans. Database Syst.*, 13(1):91–128, 1988.

[19] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. *Disc. Math. and Theor. Comp. Sci.*, AH:127–146, 2007.

[20] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[21] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[22] S. Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.

[23] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[24] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc. SPAA*, pages 281–291, 2001.

[25] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. STOC*, pages 373–380, 2004.

[26] P. Indyk and D. P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proc. FOCS*, pages 283–, 2003.

[27] D. M. Kane, J. Nelson, and D. P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proc. SODA*, pages 1161–1178, 2010.

[28] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[29] M. H. Overmars. *The Design of Dynamic Data Structures*. Springer, 1983.

[30] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, and M. Huras. Multi-dimensional clustering: A new data layout scheme in db2. In *SIGMOD*, pages 637–641, 2003.

[31] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[32] C. R. Palmer, G. Siganos, M. Faloutsos, and C. Faloutsos. The connectivity and fault-tolerance of the internet topology. In *NRDM Workshop*, 2001.

[33] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[34] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proc. VLDB*, pages 522–531, 1996.

[35] A. Siegel. On universal classes of extremely random

constant-time hash functions. *SIAM J. Computing*, 33(3):505–543, 2004.
[36] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. SODA*, pages 167–175, 2004.

# APPENDIX

# A. APPENDIX

## A.1 Expectation and variance analyses for balls and bins with limited independence

The following is a proof of Lemma 2. Below, $X$ corresponds to the random variable which is the number of bins receiving at least one ball when tossing $A$ balls into $K$ independently at random.

**Lemma 2 (restatement).** *There exists some constant $\varepsilon_0$ such that the following holds for $\varepsilon \le \varepsilon_0$. Let $A$ balls be mapped into $K$ bins using a random $h \in \mathcal{H}_{2(k+1)}([A],[K])$, where $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$ for a sufficiently large constant $c > 0$. Suppose $1 \le A \le K$. For $i \in [K]$, let $X'_i$ be an indicator variable which is 1 if and only if there exists at least one ball mapped to bin $i$ by $h$. Let $X' = \sum_{i=1}^K X'_i$. Then the following hold:*

*(1).* $|\mathbf{E}[X'] - \mathbf{E}[X]| \le \varepsilon\mathbf{E}[X]$

*(2).* $\mathbf{Var}[X'] - \mathbf{Var}[X] \le \varepsilon^2$

Proof. Let $A_i$ be the random variable number counting the number of balls in bin $i$ when picking $h \in \mathcal{H}_{2(k+1)}([A],[K])$. Define the function:

$$f_k(n) = \sum_{i=0}^{k}(-1)^i\binom{n}{i}$$

We note that $f_k(0) = 1$, $f_k(n) = 0$ for $1 \le n \le k$ and $|f_k(n)| \le \binom{n}{k+1}$ otherwise. Let $f(n) = 1$ if $n = 0$ and 0 otherwise. We now approximate $X_i$ as $1 - f_k(A_i)$. We note that this value is determined entirely by $k$-wise independence of $h$. We note that this is also

$$1 - f(A_i) \pm O\left(\binom{A_i}{k+1}\right) = X_i \pm O\left(\binom{A_i}{k+1}\right).$$

The same expression holds for the $X'_i$, and thus both $\mathbf{E}[X'_i]$ and $\mathbf{E}[X_i]$ are sandwiched inside an interval of size bounded by twice the expected error. To bound the expected error we can use $(k+1)$-independence. We have that the expected value of $\binom{A_i}{k+1}$ is $\binom{A}{k+1}$ ways of choosing $k+1$ of the balls times the product of the probabilities that each ball is in bin $i$. This is

$$\binom{A}{k+1}K^{-(k+1)} \le \left(\frac{eA}{K(k+1)}\right)^{k+1} \le \frac{A}{K}\cdot(e(k+1))^{-(k+1)},$$

with the last inequality using that $A \le K$. Thus, $|\mathbf{E}[X_i] - \mathbf{E}[X'_i]| \le \varepsilon^2 A/K$ for $k = c\log(1/\varepsilon)/\log\log(1/\varepsilon)$ for sufficiently large constant $c$. In this case $|\mathbf{E}[X] - \mathbf{E}[X']| \le \varepsilon^2 A \le \varepsilon\mathbf{E}[X]$ for $\varepsilon$ smaller than some constant since $\mathbf{E}[X] = \Omega(A)$ for $A \le K$.

We now analyze $\mathbf{Var}[X']$. We approximate $X_iX_j$ as $(1 - f_k(A_i))(1 - f_k(A_j))$. This is determined by $2k$-independence of $h$ and is equal to

$$\left(1 - f(A_i) \pm O\left(\binom{A_i}{k+1}\right)\right)\left(1 - f(A_j) \pm O\left(\binom{A_j}{k+1}\right)\right)$$
$$= X_iX_j \pm O\left(\binom{A_i}{k+1} + \binom{A_j}{k+1} + \binom{A_i}{k+1}\binom{A_j}{k+1}\right)$$

We can now analyze the error using $2(k+1)$-wise independence. The expectation of each term in the error is calculated as before, except for products of the form

$$\binom{A_i}{k+1}\binom{A_j}{k+1}.$$

The expected value of this is

$$\binom{A}{k+1, k+1}K^{-2(k+1)} \le \binom{A}{k+1}^2 K^{-2(k+1)} \le \left(\frac{eA}{K(k+1)}\right)^{2(k+1)}.$$

Thus, for $k = c'\log(K/\varepsilon)/\log\log(K/\varepsilon)$ for sufficiently large $c' > 0$, each summand in the error above is bounded by $\varepsilon^3/(6K^2)$, in which case $|\mathbf{E}[X_iX_j] - \mathbf{E}[X_iX_j]| \le \varepsilon^3/K^2$. We can also make $c'$ sufficiently large so that $|\mathbf{E}[X] - \mathbf{E}[X']| \le \varepsilon^3/K^2$. Now, we have

$$\begin{aligned}
&\mathbf{Var}[X'] - \mathbf{Var}[X] \\
&\quad \le |(\mathbf{E}[X] - \mathbf{E}[X']) + 2\sum_{i<j}(\mathbf{E}[X_iX_j] - \mathbf{E}[X'_iX'_j]) \\
&\qquad\qquad - (\mathbf{E}^2[X] - \mathbf{E}^2[X'])| \\
&\quad \le |\mathbf{E}[X] - \mathbf{E}[X']| \\
&\qquad + K(K-1)\cdot\max_{i<j}|\mathbf{E}[X_iX_j] - \mathbf{E}[X'_iX'_j]| \\
&\qquad\qquad + |\mathbf{E}^2[X] - \mathbf{E}^2[X']| \\
&\quad \le \varepsilon^3/K^2 + \varepsilon^3 + \mathbf{E}^2[X](2\varepsilon^3/K^2 + (\varepsilon^3/K^2)^2) \\
&\quad \le 5\varepsilon^3
\end{aligned}$$

which is at most $\varepsilon^2$ for $\varepsilon$ sufficiently small. $\square$

## A.2 A compact lookuptable for the natural logarithm

Lemma 7. *Let $K > 4$ be a positive integer, and write $\gamma = 1/\sqrt{K}$. It is possible to construct a lookup table requiring $O(\gamma^{-1}\log(1/\gamma))$ bits such that $\ln(1 - c/K)$ can then be computed with relative accuracy $\gamma$ in constant time for all integers $c \in [4K/5]$.*

Proof. We set $\gamma' = \gamma/15$ and discretize the interval $[1, 4K/5]$ geometrically by powers of $(1+\gamma')$. We precompute the natural algorithm evaluated at $1 - \rho/K$ for all discretization points $\rho$, with relative error $\gamma/3$, creating a table $A$ taking space $O(\gamma^{-1}\log(1/\gamma))$. We answer a query $\ln(1 - c/K)$ by outputting the natural logarithm of the closest discretization point in $A$. First, we argue that the error from this output is small enough. Next, we argue that the closest discretization point can be found in constant time.

For the error, the output is up to $(1 \pm \gamma/3)$,

$$\begin{aligned}\ln(1-(1\pm\gamma')c/K) &= \ln(1-c/K \pm \gamma' c/K)\\ &= \ln(1-c/K) \pm 5\gamma' c/K\\ &= \ln(1-c/K) \pm \gamma c/(3K).\end{aligned}$$

Using the fact that $|\ln(1-z)| \geq z/(1-z)$ for $0 < z < 1$, we have that $|\ln(1-c/K)| \geq c/(K-c) \geq c/K$. Thus,

$$\begin{aligned}(1\pm\gamma/3)(\ln(1-c/(3K)) \pm \gamma c/K) &= (1\pm\gamma/3)^2 \ln(1-c/K)\\ &= (1\pm\gamma)\ln(1-c/K).\end{aligned}$$

Now, for finding the discretization point, note we need to look up $A[\lceil \log_{1+\gamma'}(c)\rceil] = A[\lceil \log(c)/(a\gamma')\rceil]$, where $a\gamma' = \log(1+\gamma')$ (note, we can compute $\log(1+\gamma') = a\gamma'$ in preprocessing). Now, write $c = d \cdot 2^k$ where $k = \lfloor \log(c)\rfloor$ and thus $1 \leq d < 2$. We can compute $k$ in $O(1)$ time since it is the most significant bit of $c$. We know $\log_{1+\gamma'}(c) = \log(d\cdot 2^k)/(a\gamma') = k/(a\gamma') + \log(d)/(a\gamma')$. Now, the derivative of the log function in the range $[1,2)$ is sandwiched between two constants. Thus, if we discretize $[1,2)$ evenly into $O(\gamma'^{-1})$ buckets and store the log of a representative of each bucket in a lookup table $B$, we can additively $O(\gamma')$-approximate $\log(d)$ by table lookup of $B[\lfloor (d-1)/\gamma'\rfloor]$. So now we have computed

$$\begin{aligned}k/(a\gamma') &+ (\log(d) + O(\gamma'))/(a\gamma')\\ &= k/(a\gamma') + \log(d)/(a\gamma') \pm O(1).\end{aligned}$$

This $O(1)$ can be taken to be arbitrarily small, say at most $1/3$, by tuning the constant in the discretization. So we know the correct index to look at in our index table $A$ up to $\pm 1/3$; since indices are integers, we are done. □

## A.3 A Rough Estimator for $L_0$-estimation

We describe here a subroutine RoughL0Estimator which gives a constant-factor approximation to $L_0$ with probability $9/16$. First, we need the following lemma which states that when $L_0$ is at most some constant $c$, it can be computed exactly in small space. The lemma follows by picking a random prime $p = \Theta(\log(mM)\log\log(mM))$ and pairwise independently hashing the universe into $[\Theta(c^2)]$ buckets. Each bucket is a counter which tracks the sum of frequencies modulo $p$ of updates to universe items landing in that bucket. The estimate of $L_0$ is then the total number of non-zero counters, and the maximum estimate after $O(\log(1/\eta))$ trials is finally output. This gives the following.

Lemma 8. *There is an algorithm which, when given the promise that $L_0 \leq c$, outputs $L_0$ exactly with probability at least $1-\eta$ using $O(c^2 \log\log(mM))$ space, in addition to needing to store $O(\log(1/\eta))$ independently chosen pairwise independent hash functions mapping $[n]$ into $[c^2]$. The update and reporting times are $O(1)$.*

Now we describe RoughL0Estimator. We pick a function $h : [n] \to [n]$ at random from a pairwise independent family. For each $0 \leq j \leq \log(n)$ we create a substream $\mathcal{S}^j$ consisting of those $x \in [n]$ with $\mathrm{lsb}(h(x)) = j$. Let $L_0(\mathcal{S})$ denote $L_0$ of the substream $\mathcal{S}$. For each $\mathcal{S}^j$ we run an instantiation $B_j$ of Lemma 8 with $c = 141$ and $\eta = 1/16$. All instantiations share the same $O(\log(1/\eta))$ hash functions $h^1, \ldots, h^{O(\log(1/\eta))}$.

To obtain our final estimate of $L_0$ for the entire stream, we find the largest value of $j$ for which $B^j$ declares $L_0(\mathcal{S}^j) > 8$. Our estimate of $L_0$ is $\tilde{L}_0 = 2^j$. If no such $j$ exists, we estimate $\tilde{L}_0 = 1$.

Theorem 11. RoughL0Estimator *with probability at least $9/16$ outputs a value $\tilde{L}_0$ satisfying $L_0 \leq \tilde{L}_0 \leq 110L_0$. The space used is $O(\log(n)\log\log(mM))$, and the update and reporting times are $O(1)$.*

Proof. The space to store $h$ is $O(\log n)$. The $\Theta(\log(1/\eta))$ hash functions $h^i$ in total require $O(\log(1/\eta)\log n) = O(\log n)$ bits to store since $1/\eta = O(1)$. The remaining space to store a single $B^j$ for a level is $O(\log\log(mM))$ by Lemma 8, and thus storing all $B^j$ across all levels requires $O(\log(n)\log\log(mM))$ space.

As for running time, upon receiving a stream update $(x, v)$, we first hash $x$ using $h$, taking time $O(1)$. Then, we compute $\mathrm{lsb}(h(x))$, also in constant time. Now, given our choice of $\eta$ for $B^j$, we can update $B^j$ in $O(1)$ time by Lemma 8.

To obtain $O(1)$ reporting time, we again use the fact that we can compute the least significant bit of a machine word in constant time. We maintain a single machine word $z$ of at least $\log(n)$ bits and treat it as a bit vector. We maintain that the $j$th bit of $z$ is 1 iff $L_0(\mathcal{S}^j)$ is reported to be greater than 8 by $B^j$. This property can be maintained in constant time during updates. Constant reporting time then follows since finding the deepest level $j$ with greater than 8 reported elements is equivalent to computing $\mathrm{lsb}(z)$.

Now we prove correctness. Observe that $\mathbf{E}[L_0(\mathcal{S}^j)] = L_0/2^{j+1}$ when $j < \log n$ and $\mathbf{E}[L_0(\mathcal{S}^j)] = L_0/2^j = L_0/n$ when $j = \log n$. Let $j^*$ be the largest $j$ satisfying $\mathbf{E}[L_0(\mathcal{S}^j)] \geq 1$ and note that $1 \leq \mathbf{E}[L_0(\mathcal{S}^{j^*})] \leq 2$. For any $j > j^*$, $\mathbf{Pr}[L_0(\mathcal{S}^j) > 8] < 1/(8\cdot 2^{j-j^*-1})$ by Markov's inequality. Thus, by a union bound, the probability that any $j > j^*$ has $L_0(\mathcal{S}^j) > 8$ is at most $(1/8)\cdot\sum_{j-j^*=1}^{\infty} 2^{-(j-j^*-1)} = 1/4$. Now, let $j^{**} < j^*$ be the largest $j$ such that $\mathbf{E}[L_0(\mathcal{S}^j)] \geq 55$, if such a $j$ exists. Since we increase the $j$ by powers of 2, we have $55 \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})] < 110$. Note that $h$ is pairwise independent, so $\mathbf{Var}[L_0(\mathcal{S}^{j^{**}})] \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]$. For this range of $\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]$, we then have by Chebyshev's inequality that

$$\mathbf{Pr}\left[|L_0(\mathcal{S}^{j^{**}}) - \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]| \geq 3\sqrt{\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]}\right] \leq 1/9.$$

If $|L_0(\mathcal{S}^{j^{**}}) - \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]| < 3\sqrt{\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]}$, then

$$32 < 55 - 3\sqrt{55} < L_0(\mathcal{S}^{j^{**}}) < 110 + 3\sqrt{110} < 142$$

since $55 \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})] < 110$.

So far we have shown that with probability at least $3/4$, $L_0(\mathcal{S}^j) \leq 8$ for all $j > j^*$. Thus, for these $j$ the $B^j$ will estimate $L_0$ of the corresponding substreams to be at most 8, and we will not output $\tilde{L}_0 = 2^j$ for $j > j^*$. On the other hand, we know for $j^{**}$ (if it exists) that with probability at least $8/9$, $\mathcal{S}^{j^{**}}$ will have $32 < L_0(\mathcal{S}_i^{j^{**}}) < 142$. By our choice of $c = 141$ and $\eta = 1/16$ in the $B^j$, $B^{j^{**}}$ will output a value $\tilde{L}_0(\mathcal{S}_i^{j^{**}}) \geq L_0(\mathcal{S}_i^{j^{**}})/4 > 8$ with probability at least $1 - (1/9 + 1/16) > 13/16$ by Lemma 8. Thus, with probability at least $1-(3/16+1/4) = 9/16$, we output $\tilde{L}_0 = 2^j$ for some $j^{**} \leq j \leq j^*$, which satisfies $110\cdot 2^j < L_0 \leq 2^j$. If such a $j^{**}$ does not exist, then $L_0 < 55$, and 1 is a 55-approximation in this case.

□