Cardinality Estimation Meets Good-Turing

Reuven Cohen Liran Katzir Aviv Yehezkel
Department of Computer Science
Technion
Haifa 32000, Israel

October 25, 2016

Abstract

Cardinality estimation algorithms receive a stream of elements whose order might be arbitrary, with possible repetitions, and return the number of distinct elements. Such algorithms usually seek to minimize the required storage and processing at the price of inaccuracy in their output. Real-world applications of these algorithms are required to process large volumes of monitored data, making it impractical to collect and analyze the entire input stream. In such cases, it is common practice to sample and process only a small part of the stream elements. This paper presents and analyzes a generic algorithm for combining every cardinality estimation algorithm with a sampling process. We show that the proposed sampling algorithm does not affect the estimator's asymptotic unbiasedness, and we analyze the sampling effect on the estimator's variance.

1 Introduction

Consider a very long stream of elements $x_1, x_2, x_3, \ldots$, with repetitions. Finding the number n of distinct elements is a well-known problem with numerous applications. The elements might represent IP addresses of packets passing through a router [20, 23, 32], elements in a large database [29], motifs in a DNA sequence [27], or nodes of RFID/sensor networks [35]. One can easily find the exact value of n by comparing the value of a newly encountered element, x_i, to every (stored) value encountered so far. If the value of x_i has not been seen before, it is stored as well. After all of the elements are treated, the stored elements are counted. This simple approach does not scale if storage is limited, or if the computation performed for each element x_i should be minimized. In these cases, the following cardinality estimation problem should be solved:

The cardinality estimation problem

Instance: A stream of elements $x_1, x_2, x_3, \ldots$ with repetitions, and an integer m. Let n be the number of different elements, namely $n = |\{x_1, x_2, x_3, \ldots\}|$, and let these elements be $\{e_1, e_2, \ldots, e_n\}$.

Objective: Find an estimate $\widehat{n}$ of n using only m storage units, where $m \ll n$.

As an application example, $x_1, x_2, x_3, \ldots$ could be IP packets received by a server. Each packet belongs to one of n IP flows $e_1, e_2, \ldots, e_n$, and the cardinality n represents the number of active flows. By monitoring the number of distinct flows during every time period, a router can estimate the network load imposed on the end server and detect anomalies. For example, it can detect DDoS attacks on the server when the number of flows significantly increases during a short time interval [16, 23].

Several algorithms have been proposed for the cardinality estimation problem [9, 10, 18, 28, 31, 32], all of which were designed to work on the entire stream, namely, without sampling. However, real-world applications are required to process large volumes of monitored data, making it impractical to collect and process the entire stream. For example, this is the case for IP packets received over a high-speed link, because a 100 Gbps link creates a 1 TB log file in less than 1.5 minutes. In such cases, only a small part of the stream is sampled and processed [13, 14].

In this paper we present and analyze a generic algorithm that adds a sampling process into every cardinality estimation procedure. The proposed algorithm consists of two steps: (a) cardinality estimation of the sampled stream using any known cardinality estimator; (b) estimation of the sampling ratio. We show that the proposed algorithm does not affect the original estimator's asymptotic bias (accuracy), and we analyze the algorithm's effect on the estimator's variance (precision).

A typical application of the proposed algorithm is query optimizers, which need to determine the best (low-cost) plan for processing user queries. The cost of a plan is usually determined according to its CPU and I/O overhead, and can be estimated using the input/output cardinalities of each operator in the plan. For example, to estimate the CPU cost of a sort operator, the optimizer estimates the number of distinct tuples that have to be sorted. While existing cardinality estimators require processing the entire data stream, the algorithm presented in this paper is much more efficient because it needs to process only a small sample.

A naive approach to solving the cardinality estimation problem is to estimate the cardinality of the sampled stream and view it as an estimation for the cardinality of the whole (unsampled) stream. However, this approach yields poor results because it ignores the probability of elements that do not appear in the sample. For example, we simulated a stream of $n = 10,000$ distinct elements whose frequency in the stream follows uniform distribution $\sim \mathrm{U}(10^2, 10^4)$. We then sampled 0.1% of the stream and used the HyperLogLog algorithm [18] with $m = 200$ storage units to estimate the cardinality of the sample. We repeated this test 200 times, each on a different stream of $10,000$ distinct elements, and averaged the results. We found that the mean estimated cardinality is $\mathbb{E}[\widehat{n}] \approx 9,100$, which means a bias of 9%, and that the relative variance is $\mathrm{Var}\left[\frac{\widehat{n}}{n}\right] \approx 0.0552$. In contrast, our proposed algorithm computed a mean estimated cardinality of $\mathbb{E}[\widehat{n}] \approx 9,900$, namely a bias of only 1%, and a relative variance of only $\mathrm{Var}\left[\frac{\widehat{n}}{n}\right] \approx 0.0118$.

The rest of this paper is organized as follows. Section 2 discusses previous work. Section 3 presents our first algorithm (Algorithm 1) for combining the sampling process with a generic cardinality estimation procedure. In addition, this section presents an analysis of the asymptotic bias and variance of Algorithm 1. Section 4 presents our enhanced algorithm

(Algorithm 2), which uses subsampling in order to reduce the memory cost of Algorithm 1. This section also presents an analysis of the asymptotic bias and variance of Algorithm 2. Section 5 presents simulation results that validate our analysis in Sections 3 and 4. Finally, Section 6 concludes the paper.

2 Related Work

Several works address the cardinality estimation problem [9, 10, 18, 28, 31, 32] and propose statistical algorithms for solving it. These algorithms are efficient because they make only one pass on the data stream, and because they use a fixed and small amount of storage. The common approach is to use a random hash function that maps each element e_j into a low-dimensional data sketch $h(e_j)$, which can be viewed as a random variable. The hash function guarantees that $h(e_j)$ is identical for all the appearances of e_j. Thus, the existence of duplicates, i.e., multiple appearances of the same element, does not affect the value of the extreme order statistics. Let h be a hash function and $h(x_i)$ denote the hash value of x_i. Then, an order statistics estimator or a bit pattern estimator can be used to estimate the value of n. An order statistics estimator keeps the smallest (or largest) m hash values. These values are then used to estimate the cardinality [6, 10, 28, 30, 31]. A bit pattern estimator keeps the highest position of the leftmost (or rightmost) "1" bit in the binary representation of the hash values in order to estimate the cardinality [9, 18]. Recent surveys about the various methods can be found in [11, 25].

Real-world applications of cardinality estimation algorithms are required to process large volumes of monitored data, making it impractical to collect and analyze the entire input stream. In such cases, it is common practice to sample and process only a small part of the stream elements. For example, routers use sampling techniques to achieve scalability. The industry standard for packet sampling is sFlow [1], short for "sampled flow". Using a defined sampling rate N, an average of 1 out of N packets is randomly sampled. The flow samples are then sent as sFlow datagrams to a central monitoring server, which analyzes the network traffic.

Although sampling techniques provide greater scalability, they also make it more difficult to infer the characteristics of the original stream. One of the first works addressing inference from samples is the Good-Turing frequency estimation, a statistical technique for estimating the probability of encountering a hitherto unseen element in a stream, given a set of past samples. For a recent paper on the Good-Turing technique, see [21].

In [38, 39], the authors present an estimator for the cardinality and entropy of a stream using a sample whose length is $O(n/\log(n))$. Their main idea is to create a frequency fingerprint histogram of all sampled elements, and then run a linear program that approximates the real frequency distributions in the full stream. However, creating a fingerprint requires exact mapping and counting of all distinct elements in the given sample, which becomes difficult in most real-world applications. The algorithm proposed in the present paper requires significantly less processing overhead.

Several other works have addressed the problem of inference from samples. For example, the detection of heavy hitters, elements that appear many times in the stream, is studied in [5]. The authors propose to keep track of the volume of data that has not been sampled.

Then, a new element is skipped only when its effect on the estimation will "not be too large." The case where the elements are packets has also been addressed. In such cases, the heavy hitters are called elephants. The accuracy of detecting elephant flows is studied in [33] and [34]. The authors use Bayes' theorem for determining the threshold of sampled packets, which indicates whether or not a flow is an elephant in the entire stream.

Other works have dealt with exploiting protocol-level information of sampled packets in order to obtain accurate estimations of the size of flows in the network. For example, in [15] the authors present a TCP-specific method whose estimate is based on the TCP SYN flag in the sampled packets. Another method, which uses TCP sequence numbers, is presented in [36]. These methods can also be used to estimate the cardinality of the flows in the network, i.e., the number of active flows. However, both methods are limited to TCP flows. In this paper we present a generic algorithm that does not make any assumptions regarding the type of the input elements.

Related to the cardinality estimation problem is the problem of finding a uniform sample of the distinct values in the stream. Such a sample can be used for a variety of database management applications, such as query optimization, query monitoring, query progress indication and query execution time prediction [4, 7, 8]. Additional applications of the uniform sample pertain to approximate query answering, such as estimating the mean, the variance, and the quantiles over the distinct values of the query [2, 3, 26]. Several algorithms provide a uniform sample of the stream; for example, the authors of [24] show how to find such a sample in a single data pass. Several variations of this work are also proposed in [12, 19, 22]. However, all the discussed approaches require scanning the entire input stream, which is usually impractical. In this paper we present a generic algorithm that does not require a full data pass over the input stream.

3 Cardinality Estimation with Sampling

3.1 Preliminaries: Good-Turing Frequency Estimation

The Good-Turing frequency estimation technique is useful in many language-related tasks where one needs to determine the probability that a word will appear in a document.

Let $X = \{x_1, x_2, x_3, \ldots\}$ be a stream of elements, and let E be the set of all different elements $E = \{e_1, e_2, \ldots, e_n\}$, such that $x_i \in E$. Suppose that we want to estimate the probability $\pi(e_j)$ that a randomly chosen element from X is e_j. A naive approach is to choose a sample $Y = \{y_1, y_2, \ldots, y_l\}$ of l elements from X, and then to let $\pi(e_j) = \frac{\#(e_j)}{l}$, where $\#(e_j)$ denotes the number of appearances of e_j in Y. However, this approach is inaccurate, because for each element e_j that does not appear in Y even once (an "unseen element"), $\#(e_j) = 0$, and therefore $\pi(e_j) = 0$.

Let $E_i = \{e_j | \#(e_j) = i\}$ be the set of elements that appear i times in the sample Y. Thus, $\sum |E_i| \cdot i = l$. The Good-Turing frequency estimation claims that $\widehat{P}_i = (i+1)\frac{|E_{i+1}|}{l}$ is a consistent estimator for the probability P_i that an element of X appears in the sample i times.

For the special case of P_0, we get from Good-Turing that $\widehat{P_0} = |E_1|/l$. In other words, the hidden mass P_0 can be estimated by the relative frequency of the elements that appear

exactly once in the sample Y. For example, if 1/10 of the elements in Y appear only once in Y, then approximately 1/10 of the elements in X do not appear in Y at all (i.e., they are unseen elements).

3.2 The Proposed Algorithm

We now show how to use Good-Turing in order to combine a sampling process with a generic cardinality estimation procedure, referred to as Procedure 1. As before, let $X = \{x_1, x_2, x_3, \ldots\}$ be the entire stream of elements, and let $Y = \{y_1, y_2, \ldots, y_l\}$ be the sampled stream. Assume that the sampling rate is P, namely, $1/P$ of the elements of X are sampled into Y. Let n and n_s be the number of distinct elements in X and Y respectively. The algorithm receives the sampled stream Y as an input and returns an estimate for n. The algorithm consists of two steps: (a) estimating n_s using Procedure 1 (any procedure, such as in [9, 18, 28, 31]); (b) estimating n/n_s, the factor by which to multiply the cardinality n_s of the sampled stream in order to estimate the cardinality n of the full stream.

To estimate n_s in step (a), Procedure 1 is invoked using m storage units. To estimate n/n_s in step (b), we first note that N_0, i.e., the number of distinct elements that do not appear in the sample, satisfies $N_0 = n - n_s$. By the definition of P_0, $N_0 = P_0 \cdot n$ holds as well. Combining these two equalities yields that $P_0 = (n - n_s)/n$ and thus $1/(1 - P_0) = n/n_s$. Therefore, the problem of estimating n/n_s is reduced to estimating the probability P_0 of unseen elements. As indicated above, by Good-Turing, $\widehat{P_0} = |E_1|/l$ is a consistent estimator for P_0. Thus, we only need to find the number $|E_1|$ of elements that appear exactly once in the sampled stream Y. To compute the value of $|E_1|$ precisely, one should keep track of all the elements in Y and ignore each previously encountered element. This is done by Algorithm 1 below using $O(l)$ storage units. We later show (Algorithm 2 in Section 4) that the number of storage units can be reduced by estimating the value of $|E_1|/l$.

Algorithm 1
(cardinality estimation with sampling)

(a) $\widehat{n_s} \leftarrow$ *Procedure1*(Y).

(b) $\widehat{P_0} \leftarrow |E_1|/l$. *The value of* $|E_1|$ *is computed precisely and* l *is known.*

(c) $\widehat{n/n_s} \leftarrow \frac{1}{1-\widehat{P_0}}$.

(d) Return $(\widehat{n} = \widehat{n_s} \cdot \widehat{n/n_s})$.

3.3 Analysis of Algorithm 1

In this section we analyze the asymptotic bias and variance of Algorithm 1, assuming that the HyperLogLog algorithm [18] is used as Procedure 1. This algorithm is the best known cardinality estimator and it has a relative variance of $\mathrm{Var}\left[\frac{\widehat{n}}{n}\right] \approx 1.08/m$, where m is the number of used storage units. Our main result is Theorem 1, where we prove that the sampling does not affect the estimator's asymptotic unbiasedness, and we show the effect of the sampling rate P on the estimator's variance.

We start with three preliminary lemmas. The first lemma, known as the Delta Method, can be used to compute the probability distribution for a function of an asymptotically normal estimator using the estimator's variance:

Lemma 1 (Delta Method)
Let θ_m be sequence of random variables satisfying $\sqrt{m}(\theta_m - \theta) \to \mathcal{N}(0, \sigma^2)$, where θ and σ^2 are finite valued constants. Then, for every function g for which $g'(\theta)$ exists and $g'(\theta) \neq 0$, the following holds:

$$\sqrt{m}(g(\theta_m) - g(\theta)) \to \mathcal{N}\left(0, \sigma^2 g'(\theta)^2\right).$$

A proof is given in [37].

The next lemma shows how to compute the probability distribution of a random variable that is a product of two normally distributed random variables whose covariance is 0:

Lemma 2 (Product distribution)
Let X and Y be two random variables satisfying $X \to \mathcal{N}(\mu_x, \sigma_x^2)$ and $Y \to \mathcal{N}(\mu_y, \sigma_y^2)$, such that $\mathrm{Cov}[X, Y] = 0$. Then, the product $X \cdot Y$ asymptotically satisfies the following:

$$X \cdot Y \to \mathcal{N}\left(\mu_x \mu_y, \mu_y^2 \sigma_x^2 + \mu_x^2 \sigma_y^2\right).$$

The proof follows directly from applying the multivariate version of the Delta Method on the function $g(X, Y) = (X \cdot Y, 1)$.

The last lemma states a normal limit law for $|E_1|/l$, where $|E_1|$ and l are as described in Section 3.1:

Lemma 3
$|E_1|/l \to \mathcal{N}\left(P_0, \frac{1}{l}((|E_1| + 2|E_2|)/l - (|E_1|/l)^2)\right)$, *where l is the sample size.*

Proof:
Theorem 1 in [17] states that[1]

$$1 - |E_1|/l \to \mathcal{N}\left(C, \frac{1}{l}((|E_1| + 2|E_2|)/l - (|E_1|/l)^2)\right).$$

C is defined there as "the coverage", and it satisfies that "$1-C$ is equivalent to the probability that the next observation will belong to a new class". Thus, according to our notations, $1 - C = P_0$, and the previous equation is equivalent to

$$|E_1|/l \to \mathcal{N}\left(P_0, \frac{1}{l}((|E_1| + 2|E_2|)/l - (|E_1|/l)^2)\right).$$

■

We are now ready to start our analysis. Our first lemma summarizes the distribution of P_0:

[1] Notice the change in the notations: in [17] the authors use N_i instead of E_i and n instead of l.

Lemma 4
$\widehat{P_0} \to \mathcal{N}\left(P_0, \frac{1}{l}\left(P_0(1-P_0)+P_1\right)\right)$, *where l is the sample size.*

Proof:
For the expectation, the following holds

$$\mathbb{E}\left[\widehat{P_0}\right] = \mathbb{E}[|E_1|/l] = P_0.$$

The first equality is due to the definition of $\widehat{P_0}$ in Algorithm 1, and the second is due to Lemma 3.

For the variance, the following holds

$$\mathrm{Var}\left[\widehat{P_0}\right] = \mathrm{Var}\left[|E_1|/l\right] = 1/l \cdot ((|E_1| + 2|E_2|)/l - (|E_1|/l)^2).$$

The first equality is due to the definition of $\widehat{P_0}$ in Algorithm 1. The second equality is due to Lemma 3. Finally, due to Good-Turing we get that $1/l \cdot ((|E_1| + 2|E_2|)/l - (|E_1|/l)^2) \to \frac{1}{l}\left(P_0(1-P_0)+P_1\right)$. ■

As shown in [18], when sampling is not used, Procedure 1 estimates n with mean value n and variance $\frac{1.03 \cdot n^2}{m}$ (approximated in the rest of the paper as $\frac{n^2}{m}$). Therefore, using standard statistical claims, the distribution of the estimator can be approximated as the normal distribution with the abovementioned mean and variance, namely, $\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{m}\right)$. The following theorem states the asymptotic bias and variance of Algorithm 1 for $P < 1$.

Theorem 1
Algorithm 1 estimates n with mean value n and variance $\frac{n^2}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}$, namely, $\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}\right)$, where l is the sample size, and m is the storage size used for estimating n_s. In addition, P_0 and P_1 satisfy:

1. $\mathbb{E}[P_0] = \frac{1}{n}\sum_{i=1}^{n} e^{-P \cdot f_i}$.
2. $\mathbb{E}[P_1] = \frac{P}{n}\sum_{i=1}^{n} f_i \cdot e^{-P \cdot f_i}$

where f_i is the frequency of element e_i in X.

Proof:
Applying the Delta Method (Lemma 1) on $\frac{1}{1-P_0}$ yields that

$$\frac{1}{1-\widehat{P_0}} \to \mathcal{N}\left(\frac{1}{1-P_0}, \frac{1}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^4}\right). \tag{1}$$

According to [18] (see also the explanation in the paragraph preceding Theorem 1):

$$\widehat{n}_s \to \mathcal{N}\left(n_s, \frac{n_s^2}{m}\right). \tag{2}$$

Next, we show that $\frac{1}{1-P_0}$ and n_s have zero covariance:

$$\begin{aligned}
\mathrm{Cov}\left[n_s, \frac{1}{1-P_0}\right] &= \mathrm{Cov}\left[n_s, \frac{n}{n_s}\right] \\
&= \mathbb{E}\left[\mathrm{Cov}\left[n_s, \frac{n}{n_s} \mid n_s\right]\right] + \mathrm{Cov}\left[\mathbb{E}[n_s \mid n_s], \mathbb{E}\left[\frac{n}{n_s} \mid n_s\right]\right] \\
&= 0 + \mathrm{Cov}\left[n_s, \frac{n}{n_s}\right] \\
&= \mathbb{E}\left[n_s \cdot \frac{n}{n_s}\right] - \mathbb{E}[n_s]\,\mathbb{E}\left[\frac{n}{n_s}\right] \\
&= \mathbb{E}[n] - n_s \cdot \frac{n}{n_s} \\
&= n - n = 0.
\end{aligned}$$

The first equality is due to the P_0 definition. The second equality is due to the law of total covariance. The third equality is because n_s and n/n_s are independent when n_s is known. The fourth equality is due to the covariance definition. The fifth and sixth equalities are due to the expectation definition and algebraic manipulations.

Applying the distribution product property (Lemma 2) for Eqs. (1) and (2) yields that:

$$\widehat{n} = \frac{\widehat{n}_s}{1-\widehat{P_0}} \to \mathcal{N}\left(n, \frac{n_s^2}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^4} + \frac{n_s^2}{m}\frac{1}{(1-P_0)^2}\right).$$

Finally, substituting $n_s = n \cdot (1-P_0)$ yields that:

$$\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}\right).$$

The resulting asymptotic variance depends on both P_0 and P_1, which are determined according to the sampling rate P and f_i, the frequency of each distinct element in the stream. Thus, the final part of the proof is to compute their expectation. For P_0 we get that:

$$\mathbb{E}[P_0] = \frac{1}{n}\sum_{i=1}^{n}(1-P)^{f_i} = \frac{1}{n}\sum_{i=1}^{n}((1-P)^{1/P})^{P\cdot f_i} = \frac{1}{n}\sum_{i=1}^{n}(e^{-1})^{P\cdot f_i} = \frac{1}{n}\sum_{i=1}^{n}e^{-P\cdot f_i}.$$

The first equality is due to the expectation and P_0 definitions. The second and the last equalities are due to algebraic manipulations. The third equality is due to the known limit result where $(1-x)^{1/x} \to e^{-1}$ when $x \to 0$ (in our case $P \to 0$).

For P_1 we get that:

$$\mathbb{E}[P_1] = \frac{1}{n}\sum_{i=1}^{n} f_i \cdot P(1-P)^{f_i-1} = \frac{1}{n}\sum_{i=1}^{n} f_i \cdot P((1-P)^{1/P})^{P\cdot(f_i-1)} = \frac{P}{n}\sum_{i=1}^{n} f_i \cdot e^{-P\cdot f_i}.$$

The first equality is due to the expectation and P_1 definitions. The second and third equalities are due to algebraic manipulations and the same known limit result noted above. ■

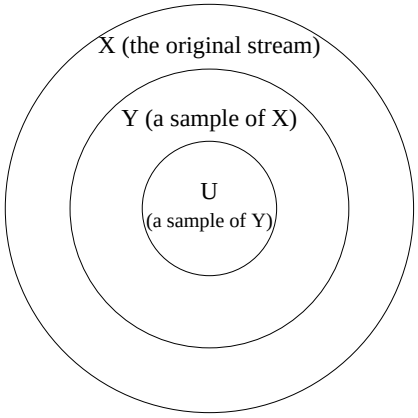

Figure 1: The relationship between X, Y and U

4 Reducing the Computational Cost of Algorithm 1

4.1 Algorithm 2 with Subsampling

Algorithm 1 computes $|E_1|$ precisely. To this end, it uses $O(l)$ storage units, which is linear in the sample size. For practical real-world applications, such as those described in Section 1, this storage size is neither practical nor scalable. In such applications, the goal is to minimize the required storage and to use small fixed-size memory. We now show how to obtain this goal by approximating the value of $|E_1|$ using a subsample U of the sample Y (see Figure 1).

Algorithm 2
(cardinality estimation with sampling and subsampling)
Same as Algorithm 1, except that in step (b) the ratio $|E_1|/l$ is estimated by invoking Procedure 2.

Procedure 2:

1. Uniformly subsample u elements from the sampled stream Y. Let this subsample be U.
2. Compute (precisely) the number $|U_1|$ of elements that appear only once in U.
3. Return $\widehat{P_0} = |U_1|/u$.

The intuition behind Algorithm 2 is that the cheap operation of Algorithm 1, estimating n_s, is performed on the whole sample Y, whose length is l, while the expensive operation, computing the number of elements that appear only once ($|E_1|$), is performed on a small subsample U of length u, where $u \ll l$.

Uniform subsampling (step (1) in Procedure 2) can be implemented using one-pass reservoir sampling [40], as follows. First, initialize U with the first u elements of Y, namely,

$y_1, y_2, \ldots, y_u$, and sort them in decreasing order of their hash values. When a new element is sampled into Y, its hash value is compared to the current maximal hash value of the elements in U. If the hash value of the new element is smaller than the current maximal hash value of U, the new value is stored in U instead of the element with the maximal hash value. After all of the elements are treated and the sample Y is created, U is a uniform subsample of length u.

We now analyze the running time complexity of Algorithm 2. Both steps (a) and (b) are performed using a simple pass over the sample Y, and require $O(1)$ operations per sampled element. Thus, these steps require $O(l)$ operations. Step (b) requires additional $O(u)$ operations for each insertion of an element into U. On the average, there are $O(\log l)$ such insertions. The total complexity is thus $O(l + u \cdot \log l) = O(l)$, which is similar to that of Algorithm 1. However, the main advantage of Algorithm 2 over Algorithm 1 is that it requires only $m + u$ storage units, while Algorithm 1 requires $m + l$ storage units, where $u \ll l$.

Next, we analyze the asymptotic bias and variance of Algorithm 2, assuming that the HyperLogLog algorithm [18] is used as Procedure 1. Then we generalize the analysis for any cardinality estimation procedure.

4.2 Analysis of Algorithm 2

Our main result is Theorem 2, which proves that the subsampling does not affect the asymptotic unbiasedness of the estimator and analyzes the effect of the sampling rate P on the estimator's variance, with respect to the storage sizes m and u.

Let Z_i be the set of elements that appear exactly i times in the subsample U; thus, $\sum |Z_i| \cdot i = u$ and Z_1 is the set of elements that appear only once in U. $|Z_1|$ can be written using indicator variables as:

$$|Z_1| = \sum_{j=1}^{u} I_j, \qquad \text{where}$$

$$I_j = \begin{cases} 1 & \text{if the j'th element in } U \text{ has a single appearance in the subsample} \\ 0 & \text{otherwise.} \end{cases}$$

Consider the estimator $\widehat{|E_1|/l}$ for $|Z_1|/u$. By definition, the variable $|Z_1|$ follows a hypergeometric distribution, which can be relaxed to a binomial distribution if $u \ll l$ [37]. Thus, due to binomial distribution properties, the expectation is

$$\mathbb{E}\left[\widehat{|E_1|/l} \;\middle|\; |E_1|\right] = \mathbb{E}[|Z_1|/u] = \mathbb{E}[I_j] = |E_1|/l, \tag{3}$$

and the variance is

$$\mathrm{Var}\left[\widehat{|E_1|/l} \;\middle|\; |E_1|\right] = \mathrm{Var}[|Z_1|/u] = 1/u \cdot \mathrm{Var}[I_j] = 1/u \cdot |E_1|/l \cdot (1 - |E_1|/l). \tag{4}$$

The following lemma summarizes the distribution of P_0:

Lemma 5
$\widehat{P_0} \to \mathcal{N}\left(P_0, \frac{1}{u}\Big(2P_0(1-P_0) + P_1\Big)\right)$.

Proof:
For the expectation, the following holds

$$\mathbb{E}\left[\widehat{P_0}\right] = \mathbb{E}\left[\widehat{|E_1|/l}\right] = \mathbb{E}\left[\mathbb{E}\left[\widehat{|E_1|/l} \mid |E_1|\right]\right] = \mathbb{E}\left[|E_1|/l\right] = P_0.$$

The first equality is due to Procedure 2. The second equality is due to the law of total expectation. The third equality is due to Eq. 3. The fourth equality is due to Lemma 3.

For the variance, the following holds:

$$\begin{aligned}
\mathrm{Var}\left[\widehat{P_0}\right] &= \mathrm{Var}\left[\widehat{|E_1|/l}\right] \\
&= \mathrm{Var}\left[\mathbb{E}\left[\widehat{|E_1|/l} \mid |E_1|\right]\right] + \mathbb{E}\left[\mathrm{Var}\left[\widehat{|E_1|/l} \mid |E_1|\right]\right] \\
&= 1/u \cdot ((|E_1| + 2|E_2|)/l - (|E_1|/l)^2) + 1/u \cdot |E_1|/l \cdot (1 - |E_1|/l) \\
&= 2/u \cdot ((|E_1| + |E_2|)/l - (|E_1|/l)^2).
\end{aligned}$$

The first equality is due to Procedure 2. The second equality is due to the law of total variance. The third equality is due to Eq. 4 and Lemma 3. The fourth equality is due to algebraic manipulations.

By Good-Turing we get that $2/u \cdot ((|E_1| + |E_2|)/l - (|E_1|/l)^2) \to \frac{1}{u}\Big(2P_0(1-P_0) + P_1\Big)$. ■

The following theorem states the asymptotic bias and variance of Algorithm 2 for $P < 1$.

Theorem 2
Algorithm 2 estimates n with mean value n and variance $\frac{n^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}$, namely, $\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}\right)$. In addition, P_0 and P_1 can be estimated as described in Theorem 1.

Proof:
Applying the Delta Method (see Section 3.3) on $\frac{1}{1-P_0}$ yields that:

$$\frac{1}{1-\widehat{P_0}} \to \mathcal{N}\left(\frac{1}{1-P_0}, \frac{1}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^4}\right). \tag{5}$$

According to [18] (see also the explanation in the paragraph preceding Theorem 1):

$$\widehat{n}_s \to \mathcal{N}\left(n_s, \frac{n_s^2}{m}\right). \tag{6}$$

Recall that $\mathrm{Cov}\left[n_s, \frac{1}{1-P_0}\right] = 0$ (see Section 3.3); applying the distribution product property (see Section 3.3) for Eqs. (5) and (6) yields that:

$$\widehat{n} = \frac{\widehat{n}_s}{1-\widehat{P_0}} \to \mathcal{N}\left(n, \frac{n_s^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^4} + \frac{n_s^2}{m}\frac{1}{(1-P_0)^2}\right).$$

Finally, substituting $n_s = n \cdot (1 - P_0)$ yields that:

$$\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{n^2}{m}\right).$$

The resulting asymptotic variance depends on both P_0 and P_1, which are determined according to the sampling rate P and f_i, the frequency of each distinct element in the stream, as was described in Section 3.3. ■

The analysis above assumes that the HyperLogLog algorithm [18] is used as Procedure 1. Recall that the asymptotic relative efficiency (ARE) of cardinality estimator $\widehat{n}$ is defined as the ratio $\text{ARE} = \frac{n^2}{m} \cdot \frac{1}{\text{Var}[\widehat{n}]}$. For example, the ARE of bottom-m sketches [28] is 1.00, and the ARE of the maximal-term sketch in [9] is 0.93. The following theorem generalizes Theorem 2 for any cardinality estimation procedure.

Theorem 3
Algorithm 2 estimates n with mean value n and variance $\frac{n^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{1}{ARE}\frac{n^2}{m}$, namely, $\widehat{n} \to \mathcal{N}\left(n, \frac{n^2}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{1}{ARE}\frac{n^2}{m}\right)$, where ARE *is the asymptotic relative efficiency of Procedure 1. In addition, P_0 and P_1 can be estimated as described in Theorem 1.*

The proof is identical to that of Theorem 2.

5 Simulation Results

This section validates our analysis for the asymptotic bias and variance of Algorithm 1 and Algorithm 2, as stated in Theorems 1 and 2 respectively. Both algorithms are implemented using the R programming language, and Procedure 1 is implemented using the HyperLogLog algorithm [18]. Then, tests are performed on synthetic data streams of n distinct elements.

Each distinct element e_j appears f_j times in the original (unsampled) stream. These frequencies are determined according to the following models:

1. Uniform distribution: The frequency of the elements is uniformly distributed between 100 and $10,000$; i.e., $f_j \sim \text{U}(10^2, 10^4)$.

2. Pareto distribution: The frequency of the elements follows the heavy-tailed rule with shape parameter α and scale parameter $s = 500$; i.e., the frequency probability function is $p(f_j) = \alpha s^\alpha f^{-\alpha-1}$, where $\alpha > 0$ and $f_j \geq s > 0$. The scale parameter s represents the smallest possible frequency.

Pareto distribution has several unique properties. In particular, if $\alpha \leq 2$, it has infinite variance, and if $\alpha \leq 1$, it has infinite mean. As α decreases, a larger portion of the probability mass is in the tail of the distribution, and it is therefore useful when a small percentage of the population controls the majority of the measured quantity.

Table 1 presents the simulation results for Algorithm 1 using uniformly distributed frequencies. The number of distinct elements is $n = 10,000$. Thus, the expected length of the original stream X is $10,000 \cdot \frac{100+10,000}{2} = 50.5 \cdot 10^6$. We examine two sampling rates:

$P = 1/100$ (Table 1(a)) and $P = 1/1000$ (Table 1(b)). We use different m values, and for every m average the results over 200 different runs. In each table row we present, for every m, the bias and the variance. The bias column is only from the simulations and it is always very close to 0, as proven in our analysis. For the variance we have two values: one from the analysis (Theorem 1) and one from the simulations.

The results in Table 1 show very good agreement between the simulation results and our analysis. First, as already said, the bias values are all very close to 0. Second, the simulation variance is always very close to the analyzed variance.

m	bias	variance	
		analysis	simulation
50	0.0023	0.0200	0.0191
100	0.0134	0.0100	0.0116
150	0.0094	0.0067	0.0057

(a) $P = 1/100$

m	bias	variance	
		analysis	simulation
50	0.0141	0.0209	0.0174
100	0.0094	0.0114	0.0099
150	0.0036	0.0096	0.0087

(b) $P = 1/1000$

Table 1: Simulation results for Algorithm 1 using uniformly distributed frequencies

Next, we consider Algorithm 2 and seek to validate Theorem 2. Table 2 presents the simulation results for uniform distribution of the frequencies. The total storage budget is 200 units, which are partitioned between m and u. The number of distinct elements is $n = 10,000$. We examine again two sampling rates: $P = 1/100$ and $P = 1/1000$. Table 3 presents results for the Pareto distribution of the frequencies, with $\alpha = 1.1$, $n = 10,000$, $P = 1/100$, and a total storage budget of $2,000$ units. The results are averaged again over 200 runs, and the variance from the analysis is determined according to Theorem 2.

m	u	bias	variance	
			analysis	simulation
10	190	0.0439	0.1000	0.1149
50	150	0.0025	0.0200	0.0217
100	100	0.0029	0.0101	0.0121
150	50	0.0037	0.0068	0.0075
190	10	0.0058	0.0060	0.0054

(a) $P = 1/100$

m	u	bias	variance	
			analysis	simulation
10	190	0.0093	0.1000	0.1081
50	150	0.0184	0.0200	0.0199
100	100	0.0114	0.0101	0.0118
150	50	0.0060	0.0068	0.0059
190	10	0.0142	0.0058	0.0053

(b) $P = 1/1000$

Table 2: Simulation results for Algorithm 2 using uniform distribution and $m + u = 200$ storage units

In both tables we see again that the bias is indeed practically 0 and that the variance of the algorithm as found by the simulations is very close to the variance found by our analysis. These results are very consistent, for both frequency distributions, both sampling rates, and all m and u values. As expected, when $m + u$ increases (more storage is used), the variance decreases.

The relative error of the variance is always less than 20% and usually less than 10%. Although our analysis is asymptotic, the simulation results show fast convergence of the

estimator to the analysis. Table 4 shows this fast convergence for uniform distribution and different values of n, where $m = 100$, $u = 1,000$ and $P = 1/100$. The results in this table are averaged over 200 runs.

m	u	bias	variance	
			analysis	simulation
50	1950	0.00005	0.0200	0.0217
100	1900	0.0189	0.0100	0.0104
500	1500	0.0011	0.0020	0.0023
1000	1000	0.00001	0.0010	0.0009
1500	500	0.0107	0.0007	0.0006

Table 3: Simulation results for Algorithm 2 using Pareto distribution and $m + u = 2000$ storage units

n	bias	variance		
		analysis	simulation	% error
500	0.0341	0.0101746	0.0083594	17.84
1,000	0.0308	0.0101744	0.0093550	8.05
10,000	0.0174	0.0101743	0.0094204	7.41
20,000	0.0158	0.0101741	0.0100634	1.09

Table 4: Simulation results for Algorithm 2 using uniform distribution showing the fast convergence rate

We now want to compare the performance of Algorithms 1 and 2. Recall that Algorithm 2 is expected to have a higher variance, but with significantly less storage. In Theorems 1 and 2 we got the following closed expressions for the relative variance of the algorithms:

1. Algorithm 1: $\frac{1}{l}\frac{P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{1}{m}$.
2. Algorithm 2: $\frac{1}{u}\frac{2P_0(1-P_0)+P_1}{(1-P_0)^2} + \frac{1}{m}$.

Recall that $m + l$ is the total storage used by Algorithm 1 (l is the sample length), and $m + u$ is the total storage used by Algorithm 2. The probabilities P_0 and P_1 are determined according to the sampling rate P and the frequency distribution of the distinct elements in the stream (see Theorem 1). Therefore, in a given stream, the only parameters that need to be determined by the user are m in Algorithm 1, and m and u in Algorithm 2. In order to find the values of m and u that yield the minimal variance for a given input stream, one only needs to know the sampling rate and then minimize the relative variance function stated above.

Table 5 presents the simulation results for $n = 10,000$, a uniform distribution of element frequencies, and for several sampling rates. Table 5(a) presents the variance of Algorithm 1. In each table row we present the sample length l, the value of m, the total storage used by the algorithm ($m+l$), and the simulation variance (averaged over 200 different runs). Recall

that in addition to m, Algorithm 1 uses $O(l)$ storage units for the exact computation of $|E_1|$. Table 5(b) presents the minimal variance of Algorithm 2 as a function of B. B indicates the total number of storage units we are willing to spend. In each table row we present the optimal partition of B between m and u that minimizes the variance of the estimator, and the simulation variance for these m and u values. For the case where $P = 1$ (no sampling), we provide in both tables the simulation variance of HyperLogLog [18], which we use as Procedure 1. This algorithm is the best known cardinality estimator and it has a relative variance of $\mathrm{Var}\left[\frac{\widehat{n}}{n}\right] \approx 1.08/m$ [18]. In this case we do not provide the values of l, m and u as there is no meaning to these parameters because sampling is not used.

P	storage			variance
	m	l	total	(simulation)
1/100	100	505,000	505,100	0.0116
	500		505,500	0.0018
	1000		506,000	0.0009
1/500	100	101,000	101,100	0.0095
	500		101,500	0.0021
	1000		102,000	0.0008
1/1000	100	50,500	50,600	0.0099
	500		51,000	0.0019
	1000		51,500	0.0008
1	100	-	100	0.0101
	500	-	500	0.0021
	1000	-	1000	0.0010

(a) Algorithm 1

P	storage			variance
	B	m	u	(simulation)
1/100	100	92	8	0.0112
	500	460	40	0.0022
	1000	921	79	0.0009
1/500	100	80	20	0.0126
	500	401	99	0.0027
	1000	803	197	0.0011
1/1000	100	72	28	0.0152
	500	363	137	0.0031
	1000	724	276	0.0013
1	100	-	-	0.0101
	500	-	-	0.0021
	1000	-	-	0.0010

(b) Algorithm 2

Table 5: Simulation results for Algorithms 1 and 2 using uniform distribution

We can easily see from the tables that the storage-variance trade-off of Algorithm 2 is *significantly better* than that of Algorithm 1. For example, the same variance (0.011) is obtained by both algorithms in the first row of $P = 1/100$. However, in this row Algorithm 1 uses 505,100 storage units whereas Algorithm 2 uses only 100. For $P = 1/500$, we see that the same variance (0.002) is obtained by the two algorithms when Algorithm 1 uses 101,500 storage units while Algorithm 2 uses only 500.

6 Conclusions

In this paper we studied the problem of estimating the number of distinct elements in a stream when only a small sample of the stream is given. We presented Algorithm 1, which combines a sampling process with a generic cardinality estimation procedure. The proposed algorithm consists of two steps: (a) cardinality estimation of the sampled stream using any known cardinality estimator; (b) estimation of the sampling ratio using Good-Turing frequency. Then we presented an enhanced algorithm that uses subsampling in order to reduce the memory cost of Algorithm 1. We proved that both algorithms do not affect the asymptotic unbiasedness of the original estimator. We also analyzed the sampling effect on the asymptotic variance of the estimators. Finally, we presented simulation results that

validate our analysis and showed how to find the optimal parameter values that yield the minimal variance.

References

[1] http://www.sflow.org.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 487–498.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD 1999*, pages 275–286.

[4] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 539–550.

[5] S. Bhattacharyya, A. Madeira, S. Muthukrishnan, and T. Ye. How to scalably and accurately skip past streams. In *ICDE 2007*, pages 654–663.

[6] P. Chassaing and L. Gérin. Efficient estimation of the cardinality of large data sets. In *Proceedings of the 4th Colloquium on Mathematics and Computer Science*, pages 419–422, 2006.

[7] S. Chaudhuri, G. Das, and V. R. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 295–306.

[8] S. Chaudhuri, G. Das, and V. R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2):9, 2007.

[9] P. Clifford and I. A. Cosma. A statistical analysis of probabilistic counting algorithms. *Scandinavian Journal of Statistics*, 2011.

[10] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *PVLDB*, 1(1):213–224, 2008.

[11] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[12] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005*, pages 25–36.

[13] N. G. Duffield. Sampling for passive internet measurement: A review. In *Statistical Science*, volume 19, pages 472–498, 2004.

[14] N. G. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement 2001*, pages 245–256.

[15] N. G. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 325–336.

[16] 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.

[17] W. W. Esty. A normal limit law for a nonparametric estimator of the coverage of a random sample. *The Annals of Statistics*, 11(3):905–912, 1983.

[18] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms (AofA) 2007*.

[19] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geometry Appl.*, 18(1/2):3–28, 2008.

[20] É. Fusy and F. Giroire. Estimating the number of active flows in a data stream over a sliding window. In *ANALCO 2007*, pages 223–231.

[21] W. A. Gale and G. Sampson. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

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

[23] S. Ganguly, M. N. Garofalakis, R. Rastogi, and K. K. Sabnani. Streaming algorithms for robust, real-time detection of ddos attacks. In *ICDCS 2007*.

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

[25] P. B. Gibbons. Distinct-values estimation over data streams. In *Data Stream Management - Processing High-Speed Data Streams*, pages 121–147. 2016.

[26] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD 1998*, pages 331–342.

[27] F. Giroire. Directions to use probabilistic algorithms for cardinality for dna analysis. *Journés Ouvertes Biologie Informatique Mathématiques*, 2006.

[28] F. Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Applied Mathematics*, 157:406–427, 2009.

[29] S. Heule, M. Nunkesser, and A. Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the EDBT 2013 Conference*.

[30] Z. B.-Y. T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, RANDOM 2002, pages 1–10.

[31] J. Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. In *Analysis of Algorithms (AofA) 2010*.

[32] A. Metwally, D. Agrawal, and A. E. Abbadi. Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT 2008, pages 618–629.

[33] T. Mori, T. Takine, J. Pan, R. Kawahara, M. Uchida, and S. Goto. Identifying heavy-hitter flows from sampled flow statistics. *IEICE Transactions*, 90-B(11):3061–3072, 2007.

[34] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement 2004*, pages 115–120.

[35] C. Qian, H. Ngan, Y. Liu, and L. M. Ni. Cardinality estimation for large-scale RFID systems. *IEEE Trans. Parallel Distrib. Syst.*, 22(9):1441–1454, 2011.

[36] B. F. Ribeiro, D. F. Towsley, T. Ye, and J. Bolot. Fisher information of sampled packets: an application to flow size estimation. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement 2006*, pages 15–26.

[37] J. Shao. *Mathematical Statistics*. Springer, 2nd edition, 2003.

[38] G. Valiant and P. Valiant. Estimating the unseen: an n/log(n)-sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 685–694, 2011.

[39] P. Valiant and G. Valiant. Estimating the unseen: Improved estimators for entropy and other properties. In *27th Annual Conference on Neural Information Processing Systems 2013*, pages 2157–2165, 2013.

[40] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.