A Unified Scheme for Generalizing Cardinality Estimators to Sum Aggregation

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

Abstract

Cardinality estimation algorithms receive a stream of elements that may appear in arbitrary order, with possible repetitions, and return the number of distinct elements. Such algorithms usually seek to minimize the required storage at the price of inaccuracy in their output. This paper shows how to generalize every cardinality estimation algorithm that relies on extreme order statistics (min/max sketches) to a weighted version, where each item is associated with a weight and the goal is to estimate the total sum of weights. The proposed unified scheme uses the unweighted estimator as a black-box, and manipulates the input using properties of the beta distribution.

1. Introduction

Consider a very long stream of elements $x_1, x_2, \ldots, x_s$ 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 [8, 9, 16], elements in a large database [12], motifs in a DNA sequence [10], or elements of RFID/sensor networks [17]. One can easily find the exact value of n in the following way. When a new element x_i is encountered, compare its value to every distinct (stored) value encountered so far. If the value of x_i has not been seen before, keep it in the storage as well. After all the elements are treated, count the number of stored elements. This simple approach does not scale if storage is limited, or if the computation performed for each element x_i should be minimized. In such a case, the following cardinality estimation problem should be solved:

The cardinality estimation problem

Instance: A stream of elements $x_1, x_2, \ldots, x_s$ with repetitions, and an integer m. Let n be the number of different elements, namely $n = |\{x_1, x_2, \ldots, x_s\}|$, 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 $m \ll n$.

Several algorithms have been proposed for the cardinality estimation problem. See [2] for a theoretical overview and [16] for a practical overview with comparative simulation results. In this paper we study the following weighted generalization of the cardinality estimation problem:

The weighted cardinality estimation problem

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

Objective: Find an estimate $\widehat{w}$ of $w = \sum_{j=1}^{n} w_j$ using only m storage units, where $m \ll n$.

An example of an instance for the cardinality estimation problem is the stream: a, b, a, c, d, b, d. For this instance, $n = |\{a, b, c, d\}| = 4$. An example of an instance for the weighted problem is: $a(3), b(4), a(3), c(2), d(3), b(4), d(3)$. For this instance, $e_1 = a, e_2 = b, e_3 = c, e_4 = d, w_1 = 3, w_2 = 4, w_3 = 2, w_4 = 3$ and $\sum w_j = 12$.

As an application example, $x_1, x_2, \ldots, x_s$ could be IP packets received by a server. Each packet belongs to one of n IP flows $e_1, e_2, \ldots, e_n$. The weight w_j can be the load imposed by flow e_j on the server. Thus, $\sum_{j=1}^{n} w_j$ represents the total load imposed on the server by all the flows to which packets $x_1, x_2, \ldots, x_s$ belong.

The main contribution of this paper is a unified scheme for generalizing any extreme order statistics estimator for the unweighted cardinality estimation problem to an estimator for the weighted cardinality estimation problem.

At first glance, it seems that a simple algorithm for the weighted version is to apply the unweighted version to each class of weights as follows: First, divide the set $\{e_1, e_2, \ldots, e_n\}$ into r pairwise disjoint subsets $E_1, E_2, \ldots, E_r$, such that all the elements of E_l have the same weight c_l, which is different from the weight c_k of any other subset E_k. Then, estimate the cardinality $\widehat{n}_l$ of every subset E_l, and return $\widehat{w} = \sum_{l=1}^{r} \widehat{n}_l c_l$. However, as shown in Section 6, for such a solution to be efficient in terms of the trade off between the required storage and the

obtained precision, one should have a priori knowledge of the distribution of the various weight classes. The unified scheme presented in this work performs better than any configuration of the aforementioned unweighted approach. Moreover, the unified scheme is agnostic to the distribution of the various weight classes.

The rest of this paper is organized as follows: In Section 2 we discuss previous works on both the unweighted and weighted cardinality estimation problems. In Section 3 we describe the beta distribution and recall several properties that will be used later. We present our new unified scheme in Sections 4 and 5. In Section 6 we compare the unified scheme with a naive algorithm, and in Section 7 we present a weighted version for several known estimators. Finally, in Section 8 we conclude the paper.

2. Related Work

State-of-the-art cardinality estimators hash every element e_j into a low dimensional data sketch $h(e_j)$, which can be viewed as a random variable (RV). The different techniques can be classified according to the data sketches they store for future processing. This paper focuses on min/max sketches [2, 6, 11, 15], which store only the minimum/maximum hashed values. The intuition behind such estimators is that each sketch carries information about the desired quantity. For example, when every element e_j is associated with a uniform RV, $h(e_j) \sim \mathrm{U}(0,1)$, the expected minimum value of $h(e_1), h(e_2), \ldots, h(e_n)$ is $1/(n+1)$. The hash function guarantees that $h(e_j)$ is identical for all the appearances of e_j. Thus, the existence of duplicates does not affect the value of the extreme order statistics. The intuition behind the new unified scheme presented in this paper is that each RV carries information about the weight of the corresponding element, and each sketch carries information about the total weight.

There are other cardinality estimation techniques other than min/max sketches. The first paper on cardinality estimation [7] describes a bit pattern sketch. In this case, the elements are hashed into a bit vector and the sketch holds the logical OR of all hashed values. Bottom-m sketches [4] are a generalization of min sketches, which maintain the m minimal values, where $m \geq 1$. Stable distribution sketches [13] generate a sketch using a vector dot product. A comprehensive overview of the different techniques is given in [2, 16].

Previous works have also dealt with the weighted problem. A statistically optimal weighted estimator for continuous variables is given in [3]. This estimator can be obtained as a direct result when the unified scheme proposed in this paper is applied to continuous max sketches. In [5], the weighted problem with integer weights is solved using binary representations. The number of storage units is not fixed, because it depends on the weights. In contrast, the proposed scheme does not assume integer weights, and uses fixed memory. Another weighted estimator, based on continuous bottom-m sketches, is given in [4]. However, bottom-m sketches require maintaining a sorted list of the bottom-m values, which is more computationally demanding than keeping the m separate minimum/maximum values, as in the proposed unified scheme.

3. The Beta Distribution

We observe that all min/max sketches can be viewed as a two step computation: (a) hash each element uniformly into $(0,1)$; and (b) store only the minimum/maximum observed value[1]. In the unified scheme we only change step (a) and hash each element into a beta distribution. The parameters of the beta distribution are derived from the weight of the element. In this section, we describe the beta distribution and two of its properties that will be used in the unified scheme.

The $\mathrm{Beta}(\alpha,\beta)$ distribution is defined over the interval $(0,1)$ and has the following probability and cumulative density functions (PDF and CDF respectively):

$$P[X = x \in (0,1)] = \frac{\Gamma(\alpha+\beta)}{\Gamma(\beta)\,\Gamma(\alpha)} x^{\alpha-1}(1-x)^{\beta-1} \quad (1)$$

$$P[X \leq x] = \int_0^x \frac{\Gamma(\alpha+\beta)}{\Gamma(\beta)\,\Gamma(\alpha)} x^{\alpha-1}(1-x)^{\beta-1} dx, \quad (2)$$

where $\Gamma(z)$ is the gamma function, defined as $\int_0^\infty e^{-t}t^{z-1}dt$. Using integration by parts, the gamma function can be shown to satisfy $\Gamma(z+1) = z \cdot \Gamma(z)$. Combining this with $\Gamma(1) = 1$ yields $\Gamma(n) = (n-1)!$ for every integer n. Two other known beta identities are [14]:

$$\mathrm{Beta}(1,1) \sim \mathrm{U}(0,1) \quad (3)$$

and

$$\mathrm{Beta}(\alpha,\beta) \sim 1 - \mathrm{Beta}(\beta,\alpha). \quad (4)$$

A key property of the beta distribution, which we use in our unified scheme is as follows:

Lemma 1.
Let $z_1, z_2, \ldots, z_n$ be independent RVs, where $z_i \sim \mathrm{Beta}(w_i, 1)$. Then,

$$\max_{i=1}^{n} z_i \sim \mathrm{Beta}\left(\sum_{i=1}^{n} w_i, 1\right).$$

Proof:

$$P[z_i \leq z] = \int_0^z w_i \cdot z^{w_i - 1} dx = z^{w_i}$$

$$P[\max_{i=1}^{n} z_i \leq z] = \prod_{i=1}^{n} P[z_i \leq z] = \prod_{i=1}^{n} z^{w_i} = z^{\sum_{i=1}^{n} w_i}.$$

[1] Some estimators (e.g. [6]) transform the uniform hashed values to induce a different distribution, and only then store the minimum/maximum observed value. In Section 7.1 we will consider such estimators.

The first equation follows by setting $\alpha = w_i$ and $\beta = 1$ in Eq. (2), and by the fact that $\Gamma(w_i + 1) = w_i\Gamma(w_i)$. The second equation is due to the multiplication of independent variables. ■

The next Lemma is a symmetric minimum version of the former one:

Lemma 2.
Let $z_1, z_2, \ldots, z_n$ be independent RVs, where $z_i \sim \text{Beta}(1, w_i)$. Then,

$$\min_{i=1}^{n} z_i \sim \text{Beta}\left(1, \sum_{i=1}^{n} w_i\right).$$

Proof:

$$\begin{aligned}\min_{i=1}^{n} z_i &= -\max_{i=1}^{n}(-z_i) \\ &= 1 - \max_{i=1}^{n}(1 - z_i) \sim 1 - \max_{i=1}^{n}\text{Beta}(w_i, 1) \\ &\sim 1 - \text{Beta}\left(\sum_{i=1}^{n} w_i, 1\right) \sim \text{Beta}\left(1, \sum_{i=1}^{n} w_i\right).\end{aligned}$$

The first and second equations are due to algebraic manipulations. The first and third distribution identities are from Eq. (4), and the second is due to Lemma 1. ■

4. The Unified Scheme

Let $x_1, x_2, \ldots, x_s$ be the values of a stream of elements with repetitions, such that $|\{x_1, x_2, \ldots, x_s\}| = n$ and $x_i \in \{e_1, e_2, \ldots, e_n\}$. Let each element e_j be associated with a weight w_j. This implies that if two elements x_{i_1} and x_{i_2} are equal (represented by the same element e_j), their weights are also equal. Let $w = \sum_{j=1}^{n} w_j$ be the value we want to estimate.

Min/max sketch estimators use a hash function to map every element x_i to $\text{U}(0, 1)$, and then remember only the minimum/maximum hashed value. If only one hash function is used, the sketch estimates the value of n with an infinite variance. To bound the variance, min/max sketches use m different hash functions in parallel and use their combined statistics for the estimation. With m hash functions, any (unweighted) min/max sketch associates each element x_i with m uniform hashed values $h_k(x_i)$, $1 \le k \le m$. The estimator remembers the minimum/maximum observed value for each hash function h_k, and uses these m values to estimate the number n of different elements. We first present this generic algorithm and then show how it can be generalized for the weighted cardinality problem. We focus on max sketch estimators, but similar algorithms can be developed for min sketches as well.

Algorithm 1.
A generic max sketch algorithm for the cardinality estimation problem

1. *Use m different hash functions, $h_1, h_2, \ldots, h_m$. For every h_k and every input element x_i, compute $h_k(x_i)$. Let $h_k^+ = \max_{i=1}^{s}\{h_k(x_i)\}$ be the maximum observed value for hash function h_k.*
2. *Invoke ProcEstimate$(h_1^+, h_2^+, \ldots, h_m^+)$ to estimate the number n of different elements.*

ProcEstimate() is the specific cardinality estimate procedure. Different algorithms employ different procedures, some of which are discussed in Section 7.2. Consider one of the hash functions h_k. Assuming that $h_k(x) \sim \text{U}(0, 1)$, then by Eq. (3) and Lemma 1 we get:

Corollary 3.
For every hash function, $h_k^+ = \max_{i=1}^{s} h_k(x_i) \sim \max_{i=1}^{n}\text{U}(0, 1) \sim \max_{i=1}^{n}\text{Beta}(1, 1) \sim \text{Beta}(n, 1)$ holds. Thus, estimating the value of n by ProcEstimate() in Algorithm 1 is equivalent to estimating the value of α in the $\text{Beta}(\alpha, 1)$ distribution of h_k^+. ■

The intuition behind our unified scheme is that instead of associating each element x_i with a uniform hashed value $h_k(x_i)$, we associate it with a RV taken from a $\text{Beta}(w_j, 1)$ distribution, where w_j is the element's weight. Technically, we first hash x_i uniformly $h_k(x_i) \sim \text{U}(0, 1)$. Then, we transform[2] $h_k(x_i)$ to $\widetilde{h}_k(x_i)$, such that $\widetilde{h}_k(x_i) \sim \text{Beta}(w_j, 1)$. Hence, and by Lemma 1, we have $\widetilde{h}_k^+ \sim \text{Beta}\left(w = \sum_{j=1}^{n} w_j, 1\right)$ instead of $h_k^+ \sim \text{Beta}(n, 1)$ as in the unweighted case. Thus, and by Corollary 3, the same algorithm that estimates n in the unweighted case can estimate w in the weighted case.

Algorithm 2.
A generic max sketch algorithm for the weighted cardinality estimation problem

1. *Use m different hash functions, $h_1, h_2, \ldots, h_m$. For every h_k and every x_i, compute $h_k(x_i) \sim \text{U}(0, 1)$ and then convert $h_k(x_i)$ into $\widetilde{h}_k(x_i)$ such that $\widetilde{h}_k(x_i) \sim \text{Beta}(w_j, 1)$. Let $\widetilde{h}_k^+ = \max_{i=1}^{s}\left\{\widetilde{h}_k(x_i)\right\}$.*
2. *Invoke ProcEstimate$(\widetilde{h}_1^+, \widetilde{h}_2^+, \ldots, \widetilde{h}_m^+)$ to estimate the value of w.*

The key point is that *ProcEstimate*() is exactly the same procedure used to estimate the unweighted cardinality n in Algorithm 1.

Lemma 4.
Estimating w by Algorithm 2 is equivalent to estimating n by Algorithm 1. Thus, Algorithm 2 estimates w with the same variance and bias as that of the underlying ProcEstimate() procedure used by Algorithm 1.

[2]We discuss the transformation from a uniform distribution to the beta distribution in Section 7.1.

Proof: From Lemma 1 follows that for every hash function h_k, $\widetilde{h}_k^+ = \max_{i=1}^{s}\left\{\widetilde{h}_k(x_i)\right\} \sim \text{Beta}\,(w,1)$. Thus, and by Corollary 3, the distribution of the input elements for *ProcEstimate*() in Algorithm 2 is exactly as the distribution of the input elements for *ProcEstimate*() in Algorithm 1: $\text{Beta}\,(\alpha,1)$, where α is the parameter to be estimated. Finally, from Corollary 3 follows that *ProcEstimate*() estimates the value of α in both cases. ■

5. Reducing the Number of Hash Functions Using Stochastic Averaging

Algorithms 1 and 2 use m different hash functions in order to obtain a better precision. In certain applications, the computational burden renders the scheme infeasible even for $m = 10$. Stochastic averaging [7] is a method to overcome the computational cost at the price of reduced statistical efficiency[3] in the estimator's variance, which is negligible in this case. The main idea is to use only two hash functions: the first for assigning a bucket (one of m) for every element x_i, and the second for associating every element x_i in every bucket with a $\text{U}(0,1)$ value[4]. The estimator then stores the maximum observed value of each bucket. We denote by H_1 the hash used for bucketing and by H_2 the hash used for generating the sketches. Formally, $H_1(x_i) \sim \text{U}\{1,2,\ldots,m\}$ and $H_2(x_i) \sim \text{U}(0,1)$. We first incorporate this concept into the generic max sketch algorithm for the cardinality estimation problem (Algorithm 1) and then into the unified scheme for the weighted cardinality estimation problem (Algorithm 2). As in the previous section, we consider max sketch estimators, but the same idea is applicable for min sketches as well.

Algorithm 3.
A generic max sketch algorithm for the cardinality estimation problem using stochastic averaging

1. *Use two hash functions:* $H_1(x_i) \sim \text{U}\{1,2,\ldots,m\}$ *and* $H_2(x_i) \sim \text{U}(0,1)$. *For every input element* x_i, *compute* $H_1(x_i)$ *and* $H_2(x_i)$. *Let* $h_k^+ = \max_{i=1}^{s}\{H_2(x_i) \mid H_1(x_i) = k\}$ *be the maximum observed value in the* k*'th bucket.*
2. *Invoke* $\mathit{ProcEstimateSA}(h_1^+, h_2^+, \ldots, h_m^+)$ *to estimate the number* n *of different elements.*

Note that *ProcEstimateSA*() in Step (2) is different from *ProcEstimate*() in Algorithm 1 and Algorithm 2. In Section 4 we defined h_k^+ to be the maximum observed value for the k'th hash, and showed in Corollary 3 that it satisfies $h_k^+ \sim \text{Beta}\,(n,1)$; namely, estimating the value of n by *ProcEstimate*() is equivalent to estimating the value of α in the $\text{Beta}\,(\alpha,1)$ distribution of h_k^+. In the following analysis we show an equivalent result for the stochastic averaging case. To this end, (a) let $\vec{h}^+ = (h_1^+, h_2^+, \ldots, h_m^+)$ be the vector of the maximum observed values in each bucket; (b) let b_k be the size of the k'th bucket, namely, $b_k = |\{H_1(x_i) = k\}|$; and (c) let $\vec{b} = (b_1, b_2, \ldots, b_m)$ be the vector of the bucket's sizes. Since h_k^+ is the maximum of b_k uniformly distributed RVs, from Lemma 1 we get:

$$\vec{h}^+ \mid \vec{b} \sim \left(\text{Beta}\,(b_1,1), \text{Beta}\,(b_2,1), \ldots, \text{Beta}\,(b_m,1)\right). \quad (5)$$

From the Central Limit Theorems follows that b_k is highly concentrated around $\frac{n}{m}$. Specifically, $b_k = \frac{n}{m} \pm O\left(\sqrt{\frac{n}{m}}\right)$. Substituting $b_k = \frac{n}{m}$ into Eq. (5) yields the following corollary, which is the stochastic averaging equivalence of Corollary 3:

Corollary 5.
For every hash function, $h_k^+ = \max_{i=1}^{s}\{H_2(x_i) \mid H_1(x_i) = k)\} \sim \text{Beta}\,(b_k,1) \approx \text{Beta}\left(\frac{n}{m},1\right)$. *Thus, estimating the value of* $\frac{n}{m}$ *by ProcEstimateSA*() *in Algorithm 3 is equivalent to estimating the value of* α *in the* $\text{Beta}\,(\alpha,1)$ *distribution of* h_k^+. ■

To generalize Algorithm 3 for solving the weighted problem using stochastic averaging, we employ the same idea used for generalizing Algorithm 1, where each element x_i is associated with a variable taken from a $\text{Beta}\,(w_j,1)$ distribution. Thus, we first use H_1 to insert each element x_i into a random bucket $1 \le k \le m$, and H_2 to associate x_i with a uniformly distributed variable $H_2(x_i) \sim \text{U}(0,1)$. Then, we transform $H_2(x_i)$ to have a beta distribution, $\widetilde{H_2}(x_i) \sim \text{Beta}\,(w_j,1)$. Hence, and by Lemma 1, we get $\widetilde{h}_k^+ \sim \text{Beta}\left(\frac{w}{m} = \sum_{\{H_1(x_i)=k\}} w_j, 1\right)$ instead of $h_k^+ \sim \text{Beta}\left(\frac{n}{m},1\right)$ in the unweighted case.

Algorithm 4.
A generic max sketch algorithm for the weighted cardinality estimation problem using stochastic averaging

1. *Use two hash functions:* $H_1(x_i) \sim \text{U}\{1,2,\ldots,m\}$ *and* $H_2(x_i) \sim \text{U}(0,1)$. *For every* x_i, *compute* $H_1(x_i)$ *and* $H_2(x_i)$, *and then convert* $H_2(x_i)$ *into* $\widetilde{H}_2(x_i)$ *such that* $\widetilde{H}_2(x_i) \sim \text{Beta}\,(w_j,1)$. *Let* $\widetilde{h}_k^+ = \max_{i=1}^{s}\left\{\widetilde{H}_2(x_i) \mid H_1(x_i) = k\right\}$ *be the maximum observed value in the* k*'th bucket.*
2. *Invoke* $\mathit{ProcEstimateSA}(\widetilde{h}_1^+, \widetilde{h}_2^+, \ldots, \widetilde{h}_m^+)$ *to estimate the value of* w.

Define $\vec{h}^+ = (\widetilde{h}_1^+, \widetilde{h}_2^+, \ldots, \widetilde{h}_m^+)$ and let $b_k = \sum_{\{H_1(x_i)=k\}} w_j$ be the sum of the elements in the k'th bucket. As for the unweighted case, when $\vec{b}$ is known, from Lemma 1 follows that:

$$\vec{h}^+ \mid \vec{b} \sim \left(\text{Beta}\,(b_1,1), \text{Beta}\,(b_2,1), \ldots, \text{Beta}\,(b_m,1)\right). \quad (6)$$

[3] See the discussion regarding "statistical efficiency" at the end of this section.

[4] In fact, it is possible to use a single hash function, as proposed in [7]. In this case, the first $\log m$ bits are used for bucketing.

Again, from the Central Limit Theorems follows that b_k is highly concentrated around $\frac{w}{m}$. Specifically, $b_k = \frac{w}{m} \pm O\left(\sqrt{\frac{1}{m}\sum_{j=1}^{n} w_j^2}\right)$. Substituting this into Eq. (6) yields $b_k \sim \text{Beta}\left(\frac{w}{m}, 1\right)$, and the following equivalence of Lemma 4:

Lemma 6.
Estimating w by Algorithm 4 is equivalent to estimating n by Algorithm 3. Thus, Algorithm 4 estimates w with the same variance and bias as that of the underlying ProcEstimateSA() procedure used by Algorithm 3.

Proof: The proof is similar to that of Lemma 4. ■

The most thorough study of the statistical effect of stochastic averaging on cardinality estimation [15] states that "it brings computational efficiency at the cost of a delayed asymptotical regime." In other words, when n is sufficiently large, the variance of each bucket size b_k is negligible. The question in turn is how large n should be in order to obtain negligible variance of b_k in the unified scheme as well. From our simulation study we deduce that for the unweighted case, when the normalized standard deviation (i.e., the variance divided by the expectation) of each b_k is less than 10^{-3}, there is negligible loss of statistical efficiency. For example, when $n = 10^6$ and $m = 1000$, we get $\text{Var}\left[b_k/\mathbb{E}[b_k]\right] \approx \frac{m}{n} = 10^{-3}$. For the weighted case, assuming that $\frac{\sum_{j=1}^{n} w_j^2}{w^2} = 10^{-6}$, we get $\text{Var}\left[b_k/\mathbb{E}[b_k]\right] = \frac{\sum_{j=1}^{n} w_j^2}{w^2} m = 10^{-3}$. However, other choices of the weights may "delay" this bound for bigger values of n. To get a more natural view of $\frac{\sum_{j=1}^{n} w_j^2}{w^2}$, let us assume that the weights w_j are drawn from a random distribution. Thereby, $w = \sum_{j=1}^{n} w_j = n\mathbb{E}[w_j]$ and $\sum_{j=1}^{n} w_j^2 = n\mathbb{E}\left[w_j^2\right]$. Using the variance definition, we obtain:

$$\frac{\sum_{j=1}^{n} w_j^2}{w^2} = \frac{n\mathbb{E}\left[w_j^2\right]}{n^2\mathbb{E}^2\left[w_j\right]} = \frac{\mathbb{E}\left[w_j^2\right] - \mathbb{E}^2\left[w_j\right] + \mathbb{E}^2\left[w_j\right]}{n\mathbb{E}^2\left[w_j\right]} = \frac{1}{n}\left(1 + \frac{\text{Var}\left[w_j\right]}{\mathbb{E}^2\left[w_j\right]}\right).$$

Therefore,

$$\text{Var}\left[b_k/\mathbb{E}[b_k]\right] = \frac{\sum_{j=1}^{n} w_j^2}{w^2} m = \frac{m}{n}\left(1 + \frac{\text{Var}\left[w_j\right]}{\mathbb{E}^2\left[w_j\right]}\right).$$

It follows that the unified scheme can deal with unbounded number of weights as long as: (1) the weights are positive; and (2) $\text{Var}\left[w_j\right]/\mathbb{E}^2\left[w_j\right]$ is a small constant.

6. A Comparison Between the Unified Scheme and a Naive Algorithm

In this section, the proposed unified scheme is compared with the following naive algorithm: (1) partition the elements into classes according to their weights; (2) estimate the cardinality of each class; (3) estimate the total weight as the weighted sum of the class cardinality estimates.

Formally, let r be the number of classes and n_l be the number of elements whose weight is c_l (i.e., $n_l = |\{e_j \mid w_j = c_l\}|$). The total weight is $w = \sum_{l=1}^{r} n_l c_l$, and the estimate made by the naive algorithm is $\widehat{w} = \sum_{l=1}^{r} \widehat{n_l} c_l$. We now show that the unified scheme outperforms this naive algorithm even if the latter has a priori knowledge about the optimal division of storage units among the $\widehat{n_l}$ estimators; namely, it makes an optimal allocation of storage units to each class, such that the variance of the total weight estimation is minimized.

The bias and variance of $\widehat{w}$ in the naive algorithm are affected by the variance of each individual estimate $\widehat{n_l}$. When an optimal *ProcEstimate*() is used both in Algorithm 2 and in the naive algorithm, e.g., the one proposed in [1], there is no bias, the variance of each individual estimate $\widehat{n_l}$ is $n_l^2/(m-2)$ and the variance of Algorithm 2 is $w^2/(m-2)$, where m is the number of hash functions or buckets. Therefore, the cardinality estimation made by the naive algorithm for each class is:

$$\widehat{n_l} \sim \mathcal{N}\left(n_l, \frac{{n_l}^2}{m-2}\right). \tag{7}$$

Let m_l be the number of storage units used to estimate n_l. The variance of the naive algorithm is therefore:

$$\text{Var}\left[\widehat{w}\right] = \text{Var}\left[\sum_{l=1}^{r} \widehat{n_l} c_l\right] = \sum_{l=1}^{r} \text{Var}\left[\widehat{n_l}\right] c_l^2 = \sum_{l=1}^{r} \frac{n_l^2 c_l^2}{m_l - 2}.$$

The first equation is due to the above definition of the naive algorithm. The second equation follows the property of the variance under linear transformation. The third equation is from Eq. (7).

An optimal choice of $(m_1, m_2, \ldots, m_r)$ would minimize $\text{Var}\left[\widehat{w}\right]$. The constraints are that $m = \sum_{l=1}^{r} m_l$ and that $\{m_1, m_2, \ldots, m_r\}$ are positive integers. We now show that even if the integrality constraints on $\{m_1, m_2, \ldots, m_r\}$ are relaxed, the variance of the unified scheme (Algorithm 2) is smaller than that of the naive algorithm. Formally, the optimal choice satisfies:

$$J\left(m_1, m_2, \ldots, m_r\right) = \sum_{l=1}^{r} \frac{n_l^2 c_l^2}{m_l - 2} \quad \text{where} \quad m = \sum_{l=1}^{r} m_l.$$

The optimal choice can be found using Lagrange multipliers:

$$\widetilde{J}\left(m_1, m_2, \ldots, m_r, \lambda\right) = \sum_{l=1}^{r} \frac{n_l^2 c_l^2}{m_l - 2} - \lambda\left(m - \sum_{l=1}^{r} m_l\right).$$

Using derivatives and solving the equation yield:

$$\frac{\partial \widetilde{J}}{\partial m_l} = -\frac{n_l^2 c_l^2}{(m_l - 2)^2} + \lambda = 0.$$

$$(m_l - 2) = n_l c_l / \sqrt{\lambda}.$$

Note that $m-2r = \sum_{l=1}^{r}(m_l-2) = \sum_{l=1}^{r}(n_l c_l)/\sqrt{\lambda} = w/\sqrt{\lambda}$. Therefore, the optimal choice is $m_l = 2+n_l c_l(m-2r)/w$ and the minimal variance is $w^2/(m-2r)$, compared to $w^2/(m-2)$ for the unified scheme.

The above analysis implies that when the number of different classes r is larger than $m/2$, or even only larger than $m/3$ if the integrality constraints are imposed, then the variance of the naive algorithm is infinite. Moreover, even if there are only a few classes, and the statistical inefficiency can be tolerated, the naive algorithm needs a priori information on the distribution of the weights, while the unified scheme does not.

7. Implementation

7.1. Transformations between distributions

In Algorithms 1, 2, 3 and 4, each element x_i is associated with a uniformly distributed RV $h(x_i) \sim \mathrm{U}(0,1)$. However, some estimators transform the uniformly distributed hashed values into another distribution. For instance, in [2, 6], $h(x_i)$ is transformed into a geometrical distribution. In this case, the transformation into a discrete distribution allows the number of bits required for storing every number to be controlled. The unified scheme proposed in this paper (Algorithms 2 and 4) transforms each hashed value $h(x_i)$ into a beta distribution $\widetilde{h}(x_i) \sim \mathrm{Beta}(w_j, 1)$. We now show how to transform a uniformly distributed RV to any distribution, and in particular to the beta distribution.

Let x be an RV whose CDF is a monotonically non-decreasing function F. The inverse function, also known as the quantile function, F^{-1} is defined as

$$F^{-1}(y) = \inf\{x : F(x) \geq y\} \quad \text{, where } 0 \leq y \leq 1. \tag{8}$$

The Inverse–Transform Method [18] is a method for generating random numbers from any probability distribution whose CDF is known, using the observation where if $u \sim \mathrm{U}(0,1)$ and $x = F^{-1}(u)$ then x has a CDF F.

Thus, to generate a random number x from distribution D with CDF F, one can generate a random number $u \sim \mathrm{U}(0,1)$ and output $x = F^{-1}(u)$. Therefore, the generic algorithms for the unweighted cardinality estimation problem, namely, Algorithm 1 and Algorithm 3, require converting each hashed value $h(x_i)$ into $\widetilde{h}(x_i) = F^{-1}(h(x_i))$. Note that the estimator may keep the original uniform hashed values without any transformation, in which case $F(x) = x$. In Table 1 we describe several transformation examples (see [14, 19] for more details).

We now return to our generic algorithms for the weighted cardinality estimation problem, namely, Algorithm 2 and Algorithm 4. The desired distribution is $\mathrm{Beta}(w_i, 1)$, whose CDF and CDF inverse are $G_{max}(x) = x^{w_j}$ and $G^{-1}_{max}(u) = u^{1/w_j}$ respectively. The Inverse–Transform Method yields that $G^{-1}_{max}(h(x_i)) = h(x_i)^{1/w_j} \sim \mathrm{Beta}(w_j, 1)$. Namely, given a uniformly distributed hashed value $h(x_i) \sim \mathrm{U}(0,1)$, taking $\widetilde{h}(x_i) = h(x_i)^{1/w_j}$ will satisfy $\widetilde{h}(x_i) \sim \mathrm{Beta}(w_j, 1)$. Thus, both Algorithms 2 and 4 require converting each hashed value $h(x_i)$ into $\widetilde{h}(x_i) = F^{-1}(G^{-1}_{max}(h(x_i)))$, where $G^{-1}_{max}(u) = u^{1/w_j}$.

source	transformation
$u \sim \mathrm{U}(0,1)$	$u^{\frac{1}{\alpha}} \sim \mathrm{Beta}(\alpha, 1)$
$u \sim \mathrm{U}(0,1)$	$(1-(1-u)^{\frac{1}{\beta}}) \sim \mathrm{Beta}(1, \beta)$
$u \sim \mathrm{U}(0,1)$	$\lfloor \log_{1-p}(1-u) \rfloor = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor \sim \mathrm{Geom}(p)$

Table 1: Distribution Transformation Examples

7.2. Examples of Specific Generalized Algorithms

We now discuss several known algorithms for the cardinality estimation problem (Problem 1), and show how each of them is extended to solve the weighted cardinality problem (Problem 2), using Algorithm 2 or Algorithm 4. For each known algorithm we present:

1. the final distribution it uses and the transformation needed from the initial uniform hashed values;
2. the *ProcEstimate*() it uses for computing the estimation; and
3. how this algorithm is extended using our unified scheme to solve the weighted cardinality estimation problem.

Continuous $\mathrm{U}(\mathbf{0},\mathbf{1})$ *with stochastic averaging*

This algorithm is presented in [1] and is the minimum variance unbiased estimator (MVUE). We summarize its key points here in brief:

1. The estimator uses uniform variables. Thus, no transformation is needed and $F^{-1}(u) = u$ holds.
2. $ProcEstimate(h_1^+, h_2^+, \ldots, h_m^+) = \frac{m(m-1)}{\sum_{k=1}^{m}(1-h_k^+)}$.
3. To generalize this estimator, we convert each hashed value into $\widetilde{H}_2(x_i) = G^{-1}_{max}(H_2(x_i))$ and then apply Algorithm 4. Thus, we get that $\widehat{w} = \frac{m(m-1)}{\sum_{k=1}^{m}(1-\widetilde{h}_k^+)}$ holds, where $\widetilde{h}_k^+ = \max\left\{(H_2(e_j))^{\frac{1}{w_j}} \mid H_1(e_j) = k\right\}$.

HyperLogLog with stochastic averaging

The algorithm presented in [6] is the state-of-the-art scheme for addressing Problem 1. We summarize its key points here in brief:

1. The estimator uses geometric random variables with success probability 1/2. Therefore, according to Table 1, $F^{-1}(u) = \lfloor -\log_2 u \rfloor \sim \mathrm{Geom}(1/2)$.
2. $ProcEstimate(\widetilde{h}_1^+, \widetilde{h}_2^+, \ldots, \widetilde{h}_m^+) = \frac{\alpha_m m^2}{\sum_{k=1}^{m} 2^{-\widetilde{h}_k^+}}$, where $\widetilde{h}_k^+ = \max\{\lfloor -\log_2(H_2(x_i)) \rfloor \mid H_1(x_i) = k\}$ and $\alpha_m = \left(m\int_0^\infty \left(\log_2\left(\frac{2+u}{1+u}\right)\right)^m du\right)^{-1}$.

3. To generalize this estimator, each hashed value is converted into $\widetilde{H}_2(x_i) = F^{-1}(G^{-1}_{max}(H_2(x_i)))$ and then Algorithm 4 is applied. Thus, we get that $\widehat{w} = \frac{\alpha_m m^2}{\sum_{k=1}^{m} 2^{-\widetilde{h}_k^+}}$ holds, where $\widetilde{h}_k^+ = \max\left\{\left\lfloor -\log_2\left(H_2(e_j)^{\frac{1}{w_j}}\right)\right\rfloor \mid H_1(e_j) = k\right\}$.

8. Conclusion

In this paper we showed how to generalize every cardinality estimation algorithm that relies on extreme order statistics (min/max sketches) to a weighted version, where each item is associated with a weight and the goal is to estimate the total sum of weights. The proposed unified scheme uses the unweighted estimator as a black box, and manipulates the input using properties of the beta distribution. We proved that estimating the weighted sum by generalizing an underlying unweighted algorithm using our unified scheme is statistically equivalent to estimating the unweighted cardinality by the underlying unweighted algorithm.

References

[1] 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*, volume AG of *Discrete Mathematics & Theoretical Computer Science Proceedings*, pages 419–422, 2006.

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

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

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

[5] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, pages 449–460, 2004.

[6] 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*. DMTCS.

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

[8] É. Fusy and F. Giroire. Estimating the number of active flows in a data stream over a sliding window. In D. Panario and R. Sedgewick, editors, *ANALCO*, pages 223–231. SIAM, 2007.

[9] S. Ganguly, M. N. Garofalakis, R. Rastogi, and K. K. Sabnani. Streaming algorithms for robust, real-time detection of ddos attacks. In *ICDCS*, page 4. IEEE Computer Society, 2007.

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

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

[12] 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*.

[13] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53:307–323, May 2006.

[14] K. Krishnamoorthy. *Handbook of Statistical Distributions with Applications*. Chapman & Hall/CRC Press, Boca Raton, FL, 2006.

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

[16] 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 '08, pages 618–629.

[17] 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.

[18] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)*. 2nd edition.

[19] C. Walck. *Handbook on Statistical Distributions for Experimentalists*. Dec. 1996.