New cardinality estimation algorithms for HyperLogLog sketches

Draft version

`https://github.com/oertl/hyperloglog-sketch-estimation-paper`

Otmar Ertl

June 25, 2016

This paper presents new methods to estimate the cardinalities of multisets recorded by HyperLogLog sketches. A theoretically motivated extension to the original estimator is presented that eliminates the bias for small and large cardinalities. Based on the maximum likelihood principle another unbiased method is derived together with a robust and efficient numerical algorithm to calculate the estimate. The maximum likelihood method is also appropriate to improve cardinality estimates of set intersections compared to the inclusion-exclusion principle. The new methods are demonstrated and verified by extensive simulations.

1. Introduction

Counting the number of distinct elements in a data stream or large datasets is a common problem in big data processing. Often there are parallel streams or data is spread over a cluster, which makes this task even more challenging. In principle, finding the number of distinct elements n with a maximum relative error ε in a data stream requires $\Omega(n)$ space [1]. However, probabilistic algorithms that achieve the requested accuracy only with high probability are able to drastically reduce space requirements. Many different probabilistic algorithms have been developed over the past two decades [2, 3]. An algorithm with an optimal space complexity of $\Omega(\varepsilon^{-2} + \log n)$ [1, 4] was finally presented [5]. This algorithm, however, is not very efficient in practice [3].

Currently, the most memory efficient algorithm that also works for distributed setups is the near-optimal HyperLogLog algorithm [6] with space complexity $\Omega(\varepsilon^{-2} \log\log n + \log n)$. The originally proposed estimation method has some problems to guarantee the same estimation error over the entire range of cardinalities. It was proposed to correct the estimate by empirical means [7, 8, 9].

In case the data is not distributed and results do not need to be aggregated further, there are even more efficient estimation algorithms available. On the one hand there

is the self-learning bitmap [10, 11], and on the other hand there is the HyperLogLog algorithm extended by a historic inverse probability estimator that is continuously updated together with the HyperLogLog sketch [3]. Both achieve the same estimation error using less space. However, the estimated cardinality depends on the insertion order of elements and hence cannot be used in a distributed environment.

2. HyperLogLog data structure

The HyperLogLog algorithm uses a sketching data structure that consists of m registers. For performance reasons the number of registers m is chosen to be a power of 2, $m = 2^p$. p is the precision that directly influences the relative error which scales like $1/\sqrt{m}$. All registers start with initial value zero. Each element insertion potentially increases the value of one of these registers. The maximum value a register can reach is a natural bound given either by the output size of the used hash algorithm or the space that is reserved for a single register. Common implementations allocate up to 8 bits per register.

2.1. Data element insertion

In order to insert a data element into a HyperLogLog data structure a hash value is calculated. The leading p bits of the hash value are used to select one of the 2^p registers. Among the next q bits, the position of the first 1-bit is determined which is a value in the range $[1, q + 1]$. The value $q + 1$ is used, if all q bits are equal to 0. If the position of the first 1-bit exceeds the value of the selected register, the register value is replaced. Algorithm 1 shows the update procedure for inserting a data element into the HyperLogLog sketch.

A HyperLogLog sketch can be characterized by the parameter pair (p, q). The first parameter controls the relative error while the second defines the possible value range of a register. A register can take all values starting from 0 to $q + 1$, inclusively.

The sum $p+q$ corresponds to the number of consumed hash value bits and defines the maximum cardinality that can be tracked. Obviously, if the cardinality reaches values in the order of 2^{p+q}, hash collisions will become more apparent and the estimation accuracy is drastically reduced.

At any time a (p, q)-HyperLogLog sketch with parameters (p, q) can be compressed into a (p', q')-HyperLogLog data structure, if $p' \leq p$ and $p' + q' \leq p + q$ is satisfied. This transformation is lossless in a sense that the resulting HyperLogLog sketch is the same as if all elements would have been recorded by a (p', q')-HyperLogLog sketch right from the beginning.

Algorithm 1 has some properties which are especially useful in distributed environments. The insertion order of elements has no influence on the final HyperLogLog sketch. Furthermore, HyperLogLog sketches with same parameters (p, q) can be easily merged. The resulting register values are simply given by the register-wise maximum values. The final HyperLogLog sketch corresponds to the union of both sets that have been represented by the two input sketches.

A $(p, 0)$-HyperLogLog sketch corresponds to a bit array as used by linear counting [12]. In this case each register value is represented by a single bit. Hence, linear counting can be regarded as a special case of the HyperLogLog algorithm for which $q = 0$.

Algorithm 1 Insertion of a data element D into a HyperLogLog data structure that consists of $m = 2^p$ registers. All register values $\vec{k} = (k_1, \ldots, k_m)$ are initially equal to 0 and remain in the range $[0, q + 1]$.

procedure InsertElement(D, $\vec{k}$)
  $\langle a_1, \ldots, a_p, b_1, \ldots, b_q \rangle_2 \leftarrow (p + q)$-bit hash value of D
  $j \leftarrow 1 + \langle a_1, \ldots, a_p \rangle_2$
  $k' = \min(\{s \mid s \in [1, q] \wedge b_s = 1\} \cup \{q + 1\})$
  $k_j \leftarrow \max(k_j, k')$
end procedure

2.2. Joint probability distribution of register values

Under the assumption of a strong universal hash function, the probability mass function for the register values $\vec{k} = (k_1, \ldots, k_m)$ of a HyperLogLog sketch with parameters p and q is given by

$$\rho(\vec{k}|n) = \sum_{n_1+\ldots+n_m=n} \binom{n}{n_1, \ldots n_m} \frac{1}{m^n} \prod_{j=1}^{m} \gamma_{n_j k_j}, \tag{1}$$

where n is the cardinality. The n distinct elements are distributed over all m registers according to a multinomial distribution with equal probabilities. $\gamma_{\nu\kappa}$ is the probability that a register is equal to κ after inserting ν distinct elements

$$\gamma_{\nu\kappa} := \begin{cases} 1 & \nu = 0 \wedge \kappa = 0 \\ 0 & \nu = 0 \wedge 1 \leq \kappa \leq q + 1 \\ 0 & \nu \geq 1 \wedge \kappa = 0 \\ \left(1 - \frac{1}{2^\kappa}\right)^\nu - \left(1 - \frac{1}{2^{\kappa-1}}\right)^\nu & \nu \geq 1 \wedge 1 \leq \kappa \leq q \\ 1 - \left(1 - \frac{1}{2^q}\right)^\nu & \nu \geq 1 \wedge \kappa = q + 1 \end{cases} \tag{2}$$

The order of register values $k_1, \ldots, k_m$ is not important for the estimation of the cardinality. More formally, the multiset $\{k_1, \ldots, k_m\}$ is a sufficient statistic for n. Since the values of the multiset are all in the range $[0, q+1]$ the multiset can also be represented as $\{k_1, \ldots, k_m\} = 0^{c_0} 1^{c_1} \cdots q^{c_q} (q + 1)^{c_{q+1}}$ where c_j is the multiplicity of value j. As a consequence, the multiplicity vector $\vec{c} := (c_0, \ldots, c_{q+1})$ is also a sufficient statistic for the cardinality. In addition, this vector contains all the information about the HyperLogLog sketch that is required for cardinality estimation. The two HyperLogLog parameters can be obtained by $p = \log_2 \|\vec{c}\|_1$ and $q = \dim \vec{c} - 2$, respectively.

2.3. Poisson approximation

Unfortunately, the probability mass function (1) is difficult to analyze, because the register values are statistically dependent. Therefore, as proposed in [6] a Poisson model can be used, which assumes that the cardinality itself is distributed according to a Poisson distribution

$$n \sim \text{Poisson}(\lambda). \tag{3}$$

Under the Poisson model the distribution of the register values is

$$\begin{aligned}
\rho(\vec{k}|\lambda) &= \sum_{n=0}^{\infty} \rho(\vec{k}|n) e^{-\lambda} \frac{\lambda^n}{n!} && (4)\\
&= \sum_{n_1=0}^{\infty} \cdots \sum_{n_m=0}^{\infty} \prod_{j=1}^{m} \gamma_{n_j k_j} e^{-\frac{\lambda}{m}} \frac{\lambda^{n_j}}{n_j! m^{n_j}} \\
&= \prod_{j=1}^{m} \sum_{n=0}^{\infty} \gamma_{n k_j} e^{-\frac{\lambda}{m}} \frac{\lambda^n}{n! m^n} \\
&= \prod_{k=0}^{q+1} \left[\sum_{n=0}^{\infty} \gamma_{nk} e^{-\frac{\lambda}{m}} \frac{\lambda^n}{n! m^n} \right]^{c_k} \\
&= e^{-c_0 \frac{\lambda}{m}} \cdot \left(\prod_{k=1}^{q} \left(e^{-\frac{\lambda}{m 2^k}} \left(1 - e^{-\frac{\lambda}{m 2^k}} \right) \right)^{c_k} \right) \cdot \left(1 - e^{-\frac{\lambda}{m 2^q}} \right)^{c_{q+1}}. && (5)
\end{aligned}$$

The final factorization shows that all register values are independent and identically distributed under the Poisson model. The probability that a register has a value less than or equal to k for a given rate λ is defined by

$$P(K \leq k|\lambda) = \begin{cases} 0 & k < 0 \\ e^{-\frac{\lambda}{m 2^k}} & 0 \leq k \leq q \\ 1 & k > q \end{cases} \tag{6}$$

2.4. Depoissonization

Due to the simpler probability mass function, it is easier to find an estimator $\hat{\lambda} = \hat{\lambda}(\vec{K})$ for the Poisson rate λ rather than for the cardinality n in the fixed-size model (1). Here $\vec{K}$ denotes the vector of all observed register values. Each component of $\vec{K}$ is distributed according to (1). Depoissonization [13] finally allows to translate the estimates back to the fixed-size model. Assume we have found an unbiased estimator for the Poisson rate

$$\mathbb{E}(\hat{\lambda}|\lambda) = \lambda \quad \text{for all } \lambda \geq 0. \tag{7}$$

We know from (4)

$$\mathbb{E}(\hat{\lambda}|\lambda) = \sum_{n=0}^{\infty} \mathbb{E}(\hat{\lambda}|n) e^{-\lambda} \frac{\lambda^n}{n!} \tag{8}$$

and therefore

$$\sum_{n=0}^{\infty} \mathbb{E}(\hat{\lambda}|n) e^{-\lambda} \frac{\lambda^n}{n!} = \lambda \tag{9}$$

holds for all $\lambda \geq 0$. The unique solution of this equation is given by

$$\mathbb{E}(\hat{\lambda}|n) = n. \tag{10}$$

Consequently, the unbiased estimator $\hat{\lambda}$ conditioned on n is also an unbiased estimator for n which motivates us to use $\hat{\lambda}$ directly as estimator for the cardinality $\hat{n} := \hat{\lambda}$. As simulation results will show later, the Poisson approximation works well over the entire cardinality range, even for estimators that are not exactly unbiased.

3. Original cardinality estimation method

The original cardinality estimator [6] is based on the idea that approximately $m2^{k-1}$ distinct element insertions are needed until the value of a register reaches k. Given that, a rough cardinality estimate can be obtained by calculating the average over the values $\{m2^{k_1-1}, \ldots, m2^{k_m-1}\}$.

In the history of the HyperLogLog algorithm different averaging techniques have been proposed. First, there was the LogLog algorithm using the geometric mean and the SuperLogLog algorithm that enhanced the estimate by truncating the largest register values before applying the geometric mean [14]. Finally, the harmonic mean was found to give even better estimates as it is inherently less sensitive to outliers. The result is the so-called raw estimator and given by

$$\hat{n}_{\text{raw}} = 2\alpha_m \,\text{HM}\Big(m2^{k_1-1}, \ldots, m2^{k_m-1}\Big) = \frac{\alpha_m m^2}{\sum_{j=1}^{m} 2^{-k_j}} = \frac{\alpha_m m^2}{\sum_{k=0}^{q+1} c_k 2^{-k}}. \tag{11}$$

Here $2\alpha_m$ is a bias correction factor. α_m for a given number of registers m was derived to be [6]

$$\alpha_m := \left(m \int_0^{\infty} \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}. \tag{12}$$

Numerical approximations of α_m for various values of m have been listed in [6]. These approximations are used in many HyperLogLog implementations. However, since the published constants have been rounded to 4 significant digits, these approximations even introduce some bias for very high precisions p. For HyperLogLog sketches that are used in practice with 256 or more registers ($p \geq 8$), it is completely sufficient to use

$$\alpha_\infty := \lim_{m\to\infty} \alpha_m = \frac{1}{2\log 2} \approx 0.7213475, \tag{13}$$

as approximation for α_m in (11), because the additional bias is negligible compared to the estimation error.

Figure 1 shows the distribution of the relative error for the raw estimator as function of the cardinality. The chart is based on 10 000 randomly generated HyperLogLog sketches.

More details of the experimental setup will be explained later in Section 3.5. Obviously, the raw estimator is biased for small and large cardinalities and fails to return accurate estimates. Therefore, to cover the entire range of cardinalities, corrections for small and large cardinalities have been proposed, respectively.

As mentioned in Section 2.1, a HyperLogLog sketch with parameters (p, q) can be turned into a $(p, 0)$-HyperLogLog sketch. Since $q = 0$ corresponds to linear counting and the transformed HyperLogLog sketch corresponds to a bitset with c_0 zeros, the linear counting cardinality estimator [12] can be applied here

$$\hat{n}_{\text{small}} = m \log(m/c_0). \tag{14}$$

The corresponding relative estimation error as depicted in Figure 2 shows that this estimator is convenient for small cardinalities. It was proposed to use this small-range estimator as long as $\hat{n}_{\text{raw}} \leq \frac{5}{2}m$ where the factor $\frac{5}{2}$ was empirically determined [6].

For large cardinalities in the order of 2^{p+q}, for which a lot of registers are already in a saturated state meaning that they have reached the maximum possible value $q + 1$, the raw estimator underestimates the cardinalities. For the 32-bit hash value case ($p+q = 32$) which was considered in [6], following correction formula was proposed to take these saturated registers into account

$$\hat{n}_{\text{large}} = -2^{32} \log\left(1 - \hat{n}_{\text{raw}}/2^{32}\right). \tag{15}$$

The original estimation algorithm as presented in [6] including small and large range corrections is summarized by Algorithm 2. The relative estimation error for the original

Algorithm 2 Original cardinality estimation algorithm for HyperLogLog sketches that use 32-bit hash values ($p + q = 32$) for insertion of data items [6].

function EstimateCardinality($\vec{k}$)
  $m \leftarrow \dim \vec{k}$
  $\hat{n}_{\text{raw}} = \alpha_m m^2 \left(\sum_{j=1}^{m} 2^{-k_j}\right)^{-1}$ ▷ raw estimate (11)
  if $\hat{n}_{\text{raw}} \leq \frac{5}{2}m$ **then**
    $c_0 = |\{j | k_j = 0\}|$
    if $c_0 \neq 0$ **then**
      return $m \log(m/c_0)$ ▷ small range correction (14)
    else
      return $\hat{n}_{\text{raw}}$
    end if
  else if $\hat{n}_{\text{raw}} \leq \frac{1}{30}2^{32}$ **then**
    return $\hat{n}_{\text{raw}}$
  else
    return $-2^{32} \log(1 - \hat{n}_{\text{raw}}/2^{32})$ ▷ large range correction (15)
  end if
end function

method is shown in Figure 3. Unfortunatley, as can be clearly seen, the ranges where the

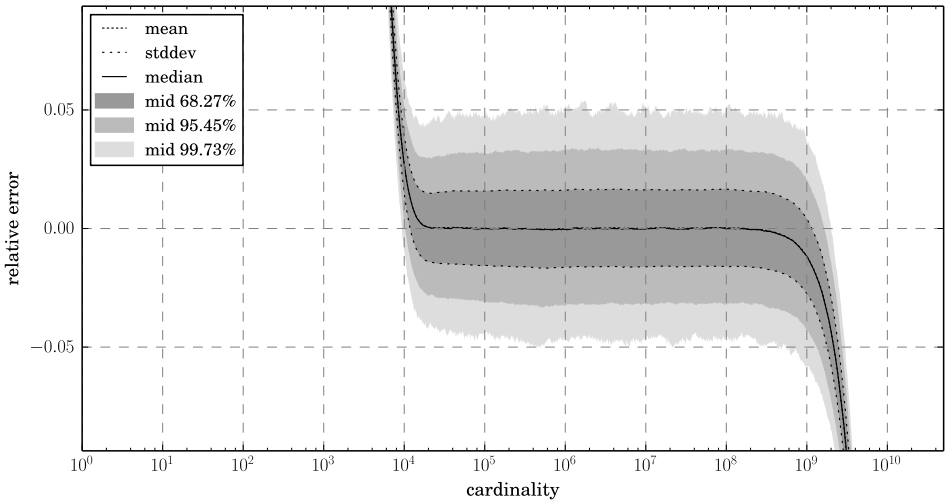

Figure 1: The distribution of the relative estimation error over the cardinality as for the raw estimator [6]. 10 000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$ have been evaluated.

estimation error is small for $\hat{n}_{\text{raw}}$ and $\hat{n}_{\text{small}}$ do not overlap. Therefore, the estimation error is much larger near the transition region. To reduce the estimation error for cardinalities close to this region, it was proposed to correct $\hat{n}_{\text{raw}}$ for bias. Empirically collected bias correction data is either stored as set of interpolation points [7], as lookup table [8], or as best-fitting polynomial [9]. However, all these empirical approaches treat the symptom and not the cause.

The large range correction is not satisfying either as it does not reduce the estimation error. Quite the contrary, it even increases the estimation error. However, instead of underestimating the cardinalities, they are now overestimated. Another indication for the incorrectness of the proposed large range correction (15) is the fact that it is not even defined for all possible states. For instance, consider a (p, q)-HyperLogLog sketch with $p + q = 32$ for which all registers are equal to the maximum possible value $q + 1$. The raw estimate would be $\hat{n}_{\text{raw}} = \alpha_m 2^{33}$, which is greater than 2^{32} and outside of the domain of the large range correction.

A simple approach to avoid the need of any large range correction is to extend the operating range of the raw estimator to larger cardinalities. This can be accomplished by using hash values with more bits ($p+q > 32$). The drawback is that 5-bit registers are no longer sufficient to represent all possible values in the range $[0, q + 1]$ in case $q \geq 30$. For HyperLogLog sketches with 6-bit registers in combination with 64-bit hash values ($p + q = 64$) as proposed in [7], it is unrealistic in practice to encounter cardinalities of order 2^{64} for which the raw estimator would return incorrect results.

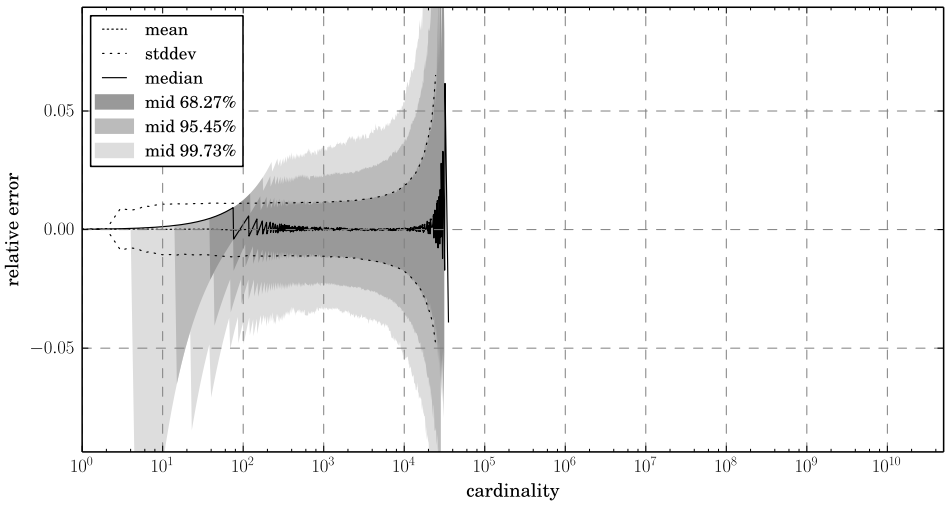

Figure 2: The distribution of the relative estimation error for the linear counting estimator which can be used for small cardinalities.

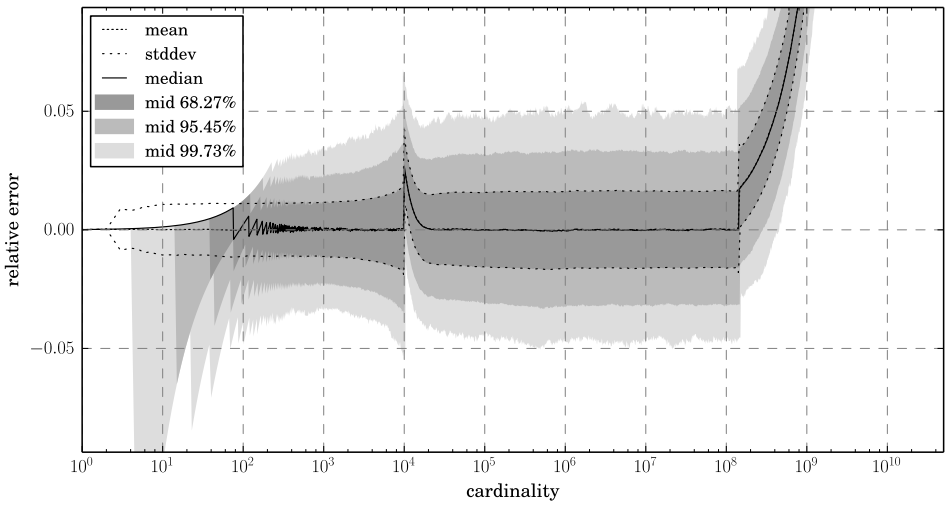

Figure 3: The distribution of the relative estimation error over the cardinality as obtained by the original estimation method [6]. 10 000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$ have been evaluated.

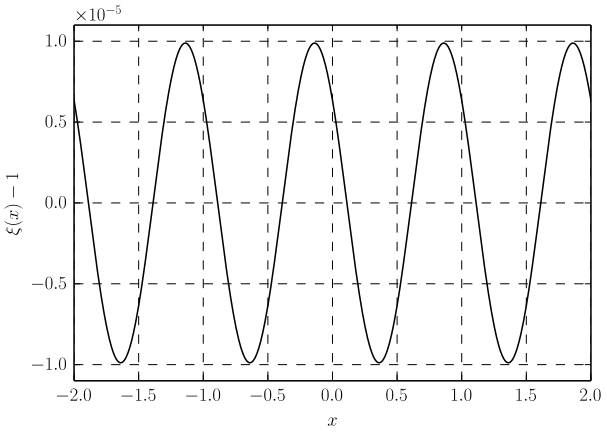

Figure 4: The deviation of $\xi(x)$ from 1.

3.1. Derivation of the raw estimator

In order to better understand why the raw estimator fails for small and large cardinalities, we start with a brief derivation that does not require complex analysis.

Consider following integer distribution

$$P(K \leq k|\lambda) = e^{-\frac{\lambda}{m2^k}}. \tag{16}$$

For now we ignore that this distribution has infinite support and differs from the register value distribution under the Poisson model (6), whose support is limited to the range $[0, q+1]$. The expectation of 2^{-k} is given by

$$\mathbb{E}(2^{-k}) = \sum_{k=-\infty}^{\infty} 2^{-k} e^{-\frac{\lambda}{m}2^{-k}} \left(1 - e^{-\frac{\lambda}{m}2^{-k}}\right) = \frac{1}{2} \sum_{k=-\infty}^{\infty} 2^{k} e^{-\frac{\lambda}{m}2^{k}} = \frac{\alpha_\infty \, m \, \xi(\log_2(\lambda/m))}{\lambda}, \tag{17}$$

where the function

$$\xi(x) := \log(2) \sum_{k=-\infty}^{\infty} 2^{k+x} e^{-2^{k+x}} \tag{18}$$

is a smooth periodic function with period 1. Numerical evaluations indicate that this function can be bounded by $1 - \varepsilon_\xi \leq \xi(x) \leq 1 + \varepsilon_\xi$ with $\varepsilon_\xi := 9.885 \cdot 10^{-6}$ (see Figure 4).

Let $k_1, \ldots, k_m$ be a sample taken from (16). For large sample sizes $m \to \infty$ we have asymptotically

$$\mathbb{E}\left(\frac{1}{2^{-k_1} + \ldots + 2^{-k_m}}\right) \underset{m\to\infty}{=} \frac{1}{\mathbb{E}(2^{-k_1} + \ldots + 2^{-k_m})} = \frac{1}{m\mathbb{E}(2^{-k})}. \tag{19}$$

Together with (17) we obtain

$$\lambda = \mathbb{E}\left(\frac{\alpha_\infty \, m^2 \, \xi(\log_2(\lambda/m))}{2^{-k_1} + \ldots + 2^{-k_m}}\right) \quad \text{for } m \to \infty. \tag{20}$$

Therefore,

$$\hat{\lambda} = \frac{\alpha_\infty \, m^2}{2^{-k_1} + \ldots + 2^{-k_m}} \tag{21}$$

is an asymptotically almost unbiased estimator for the Poisson parameter. Its asymptotic relative bias is bounded by ε_ξ. This estimator corresponds to the raw estimator (11), if the Poisson parameter estimate is used as cardinality estimate (see Section 2.3).

The estimator (21) can also be written as

$$\hat{\lambda} = \frac{\alpha_\infty \, m^2}{\sum_{k=-\infty}^{\infty} c'_k 2^{-k}}, \tag{22}$$

if c'_k is the multiplicity of value k in the sample $\{k_1, \ldots, k_m\}$. By definition, we have $\sum_{k=-\infty}^{\infty} c'_k = m$.

3.2. Limitations of the raw estimator

As we have already seen in Figure 1, the raw estimator does not work very well for small or large cardinalities. The reason is that the distribution of register values (6) differs from (16) for which the raw estimator was derived. Since a random variable K' that obeys (16) can be transformed into random variable K that follows (6) using the transformation $K = \min\left(\max\left(K', 0\right), q+1\right)$, the register values of a HyperLogLog sketch can be seen as the result after applying this transformation to a sample that is distributed according to (16). The corresponding multiplicities have following relationships

$$\begin{aligned} c_0 &= \textstyle\sum_{k=-\infty}^{0} c'_k, \\ c_k &= c'_k, \quad 1 \le k \le q, \\ c_{q+1} &= \textstyle\sum_{k=q+1}^{\infty} c'_k. \end{aligned} \tag{23}$$

Obviously, as long as $c_0 \ll m$ and $c_{q+1} \ll m$ the set of register values is similar to the originating sample. In case $c_0 = 0$ and $c_{q+1} = 0$, which implies $c'_k = 0$ for $k \le 0$ and $k \ge q+1$, we can even be sure that the register values are identical to the originating sample. This explains why the raw estimator only works well in the intermediate cardinality range for which c_0 and c_{q+1} are negligible.

3.3. Corrections to the raw estimator

Given a HyperLogLog sketch with multiplicity vector $\vec{c} = (c_0, \ldots, c_{q+1})$, we try to find estimates $\hat{c}'_k$ for c'_k for all $k \in \mathbb{Z}$ and replace them in the estimation formula (22). For $k \in [1, q]$ where $c_k = c'_k$ we can use the trivial estimators

$$\hat{c}'_k := c_k, \quad 1 \le k \le q. \tag{24}$$

To get estimators for all other $k \notin [1, q]$, we consider their expectations

$$\mathbb{E}(c'_k) = m e^{-\frac{\lambda}{m 2^k}} \left(1 - e^{-\frac{\lambda}{m 2^k}}\right). \tag{25}$$

From (6) we know that $\mathbb{E}(c_0/m) = e^{-\frac{\lambda}{m}}$ and $\mathbb{E}(1 - c_{q+1}/m) = e^{-\frac{\lambda}{m2^q}}$, and therefore, we can also write

$$\mathbb{E}(c'_k) = m\left(\mathbb{E}(c_0/m)\right)^{2^{-k}}\left(1 - \left(\mathbb{E}(c_0/m)\right)^{2^{-k}}\right) \tag{26}$$

and

$$\mathbb{E}(c'_k) = m\left(\mathbb{E}(1 - c_{q+1}/m)\right)^{2^{q-k}}\left(1 - \left(\mathbb{E}(1 - c_{q+1}/m)\right)^{2^{q-k}}\right), \tag{27}$$

which motivates us to use

$$\hat{c}'_k := m\left(c_0/m\right)^{2^{-k}}\left(1 - \left(c_0/m\right)^{2^{-k}}\right) \tag{28}$$

as estimator for $k \leq 0$ and

$$\hat{c}'_k := m\left(1 - c_{q+1}/m\right)^{2^{q-k}}\left(1 - \left(1 - c_{q+1}/m\right)^{2^{q-k}}\right) \tag{29}$$

as estimator for $k \geq q + 1$, respectively.

Inserting all these estimators into (22) as replacements for c'_k finally gives

$$\hat{\lambda} = \frac{\alpha_\infty m^2}{\sum_{k=-\infty}^{\infty} \hat{c}'_k 2^{-k}} = \frac{\alpha_\infty m^2}{m\,\sigma(c_0/m) + \sum_{k=1}^{q} c_k 2^{-k} + m\,\tau(1 - c_{q+1}/m)2^{-q}} \tag{30}$$

which we call the corrected raw estimator. Here $m\,\sigma(c_0/m)$ and $2m\,\tau(1 - c_{q+1}/m)$ are replacements for c_0 and c_{q+1} in the raw estimator (11), respectively. The functions σ and τ are defined as

$$\sigma(x) := x + \sum_{k=1}^{\infty} x^{2^k} 2^{k-1} \tag{31}$$

and

$$\tau(x) := \sum_{k=1}^{\infty} x^{2^{-k}}\left(1 - x^{2^{-k}}\right) 2^{-k}. \tag{32}$$

Using the identity $\sigma(x) + \tau(x) = \alpha_\infty \xi(\log_2(\log(1/x)))/\log(1/x)$, we get for the linear counting case with $q = 0$

$$\hat{\lambda} = \frac{\alpha_\infty m}{\sigma(c_0/m) + \tau(c_0/m)} = \frac{m \log(m/c_0)}{\xi(\log_2(\log(m/c_0)))} \approx m \log(m/c_0)\,, \tag{33}$$

which is almost identical to the linear counting estimator used for the small range correction (14). Here we used the fact that ξ can be well approximated by 1 (see Section 3.1).

3.4. Corrected raw estimation algorithm

The corrected raw estimator (30) leads to a new cardinality estimation algorithm for HyperLogLog sketches. Algorithm 3 demonstrates the numerical evaluation of the estimator. The values of functions σ and τ can be either calculated on-demand or precalculated. The possible value range for c_0 and c_{q+1} is $[0, m]$. Therefore, if performance matters, it is possible to precalculate the function values for all possible arguments and keep them in lookup tables of size $m + 1$.

Algorithm 3 Cardinality estimation algorithm based on the corrected raw estimator.

function EstimateCardinality($\vec{c}$)
  $m \leftarrow \|\vec{c}\|_1$
  $z \leftarrow \cdot m \cdot \tau(1 - c_{q+1}/m)$ ▷ alternatively, take $m \cdot \tau(1 - c_{q+1}/m)$ from precalculated lookup table
  for $k \leftarrow q, 1$ **do**
    $z \leftarrow 0.5 \cdot (z + c_k)$
  end for
  $z \leftarrow z + m \cdot \sigma(c_0/m)$ ▷ alternatively, take $m \cdot \sigma(c_0/m)$ from precalculated lookup table
  return $m^2 / (2 \log(2) z)$
end function

function $\sigma(x)$ ▷ $x \in [0, 1]$
  if $x = 1$ **then**
    return ∞
  end if
  $y \leftarrow 1$
  $z \leftarrow x$
  repeat
    $x \leftarrow x \cdot x$
    $z' \leftarrow z$
    $z \leftarrow z + x \cdot y$
    $y \leftarrow 2 \cdot y$
  until $z = z'$
  return z
end function

function $\tau(x)$ ▷ $x \in [0, 1]$
  $y \leftarrow 1$
  $z \leftarrow 0$
  repeat
    $x \leftarrow \sqrt{x}$
    $z' \leftarrow z$
    $y \leftarrow 0.5 \cdot y$
    $z \leftarrow z + (1 - x) \cdot x \cdot y$
  until $z = z'$
  return z
end function

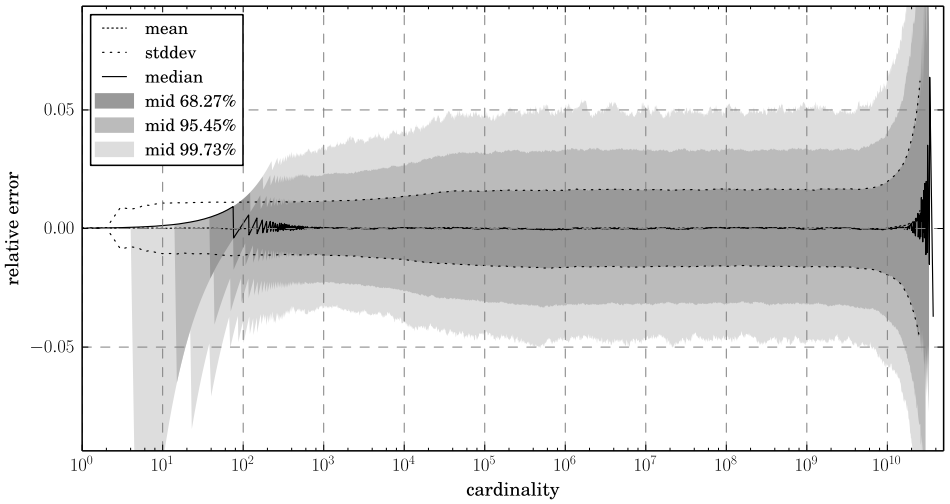

Figure 5: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

3.5. Estimation error

In order to verify the new estimation algorithm, we generated 10 000 HyperLogLog sketches and inserted up to 50 billion unique elements. Assuming a strong universal hash function, element hash values can be mocked by random numbers. For the following results we used the Mersenne Twister random number generator with 19937 bit state size from the C++ standard library.

Figure 5 shows the distribution of the relative error of the estimated cardinality using Algorithm 3 compared to the true cardinality for $p = 12$ and $q = 20$. As the mean shows, the error is unbiased over the entire cardinality range. The new approach is able to accurately estimate cardinalities up to 4 billions ($\approx 2^{p+q}$) which is about an order of magnitude larger than the operating range upper bound of the raw estimator (Figure 1) or the original method (Figure 3).

The new estimation algorithm also works well for other HyperLogLog configurations. First we considered configurations using a 32-bit hash function ($p+q = 32$). The relative estimation error for precisions $p = 8$, $p = 16$, $p = 22$ are shown in Figure 6, Figure 7, and Figure 8, respectively. As expected, since $p+q = 32$ is kept constant, the operating range remains more or less the same, while the relative error decreases with increasing precision. Again, the new algorithm gives essentially unbiased estimates. Only for very large precisions, an oscillating bias becomes apparent (compare Figure 8), that is caused by approximating the periodic function ξ by a constant (see Section 3.1).

As proposed in [7], the operating range can be extended by replacing the 32-bit hash function by a 64-bit hash function. Figure 9 shows the relative error for such a Hyper-

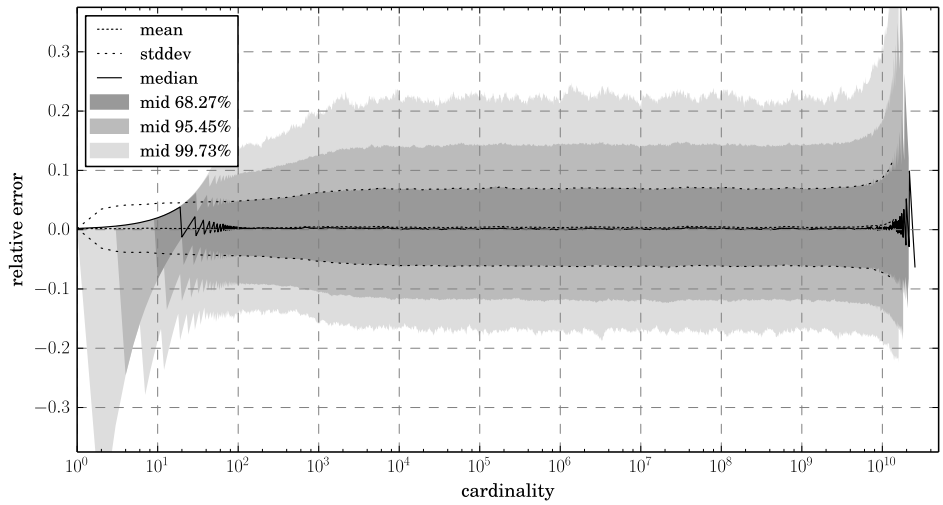

Figure 6: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

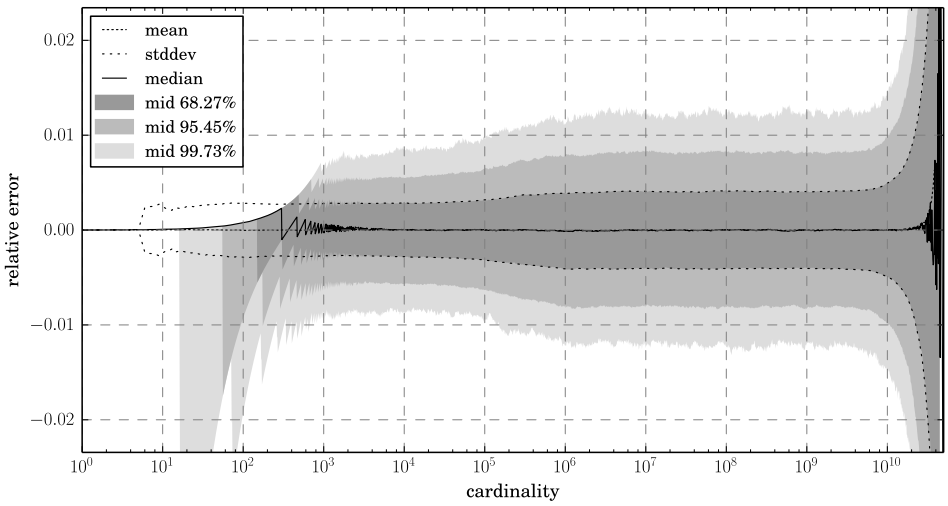

Figure 7: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

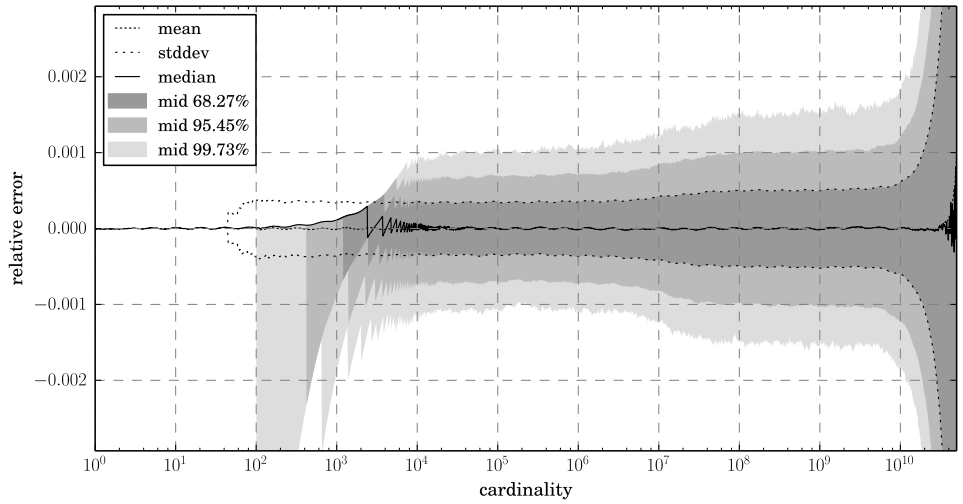

Figure 8: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 10$.

LogLog configuration with precision $p = 12$. In this case, in order to use all the 64 bits of the hash value, q must be chosen to be equal to $64 - p = 52$. As a consequence, in order to represent the maximum possible register value $q + 1 = 53$, 6 bits are needed for each register. The doubled hash value size shifts the maximum trackable cardinality value towards 2^{64}. As Figure 9 shows, when compared to the 32-bit hash value case given in Figure 5, the estimation error remains constant over the entire simulated cardinality range up to 50 billions.

We also evaluated the case $p = 12$ and $q = 14$, which is interesting, because the register values are limited to the range $[0, 15]$. As a consequence, 4 bits are sufficient for representing a single register value. This allows two registers to share a single byte, which is beneficial from a performance perspective. Nevertheless, this configuration still allows the estimation of cardinalities up to 100 millions as shown in Figure 10, which could be enough for many applications.

3.6. Performance

In order to evaluate the performance of the corrected raw estimation algorithm, we investigated the average computation time for estimating the cardinality from a given HyperLogLog sketch. For different cardinalities we loaded precalculated multiplicity vectors of 1000 randomly generated HyperLogLog sketches into main memory. The average computation time was determined by cycling over these multiplicity vectors and passing them as input to the algorithm. For each evaluated cardinality value the average execution time was calculated after 100 cycles which corresponds to 100 000 algorithm

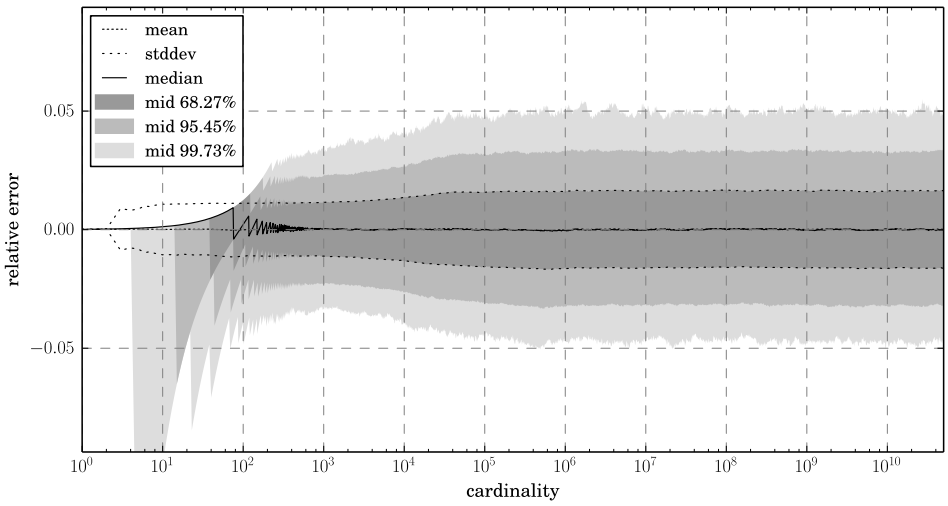

Figure 9: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 52$.

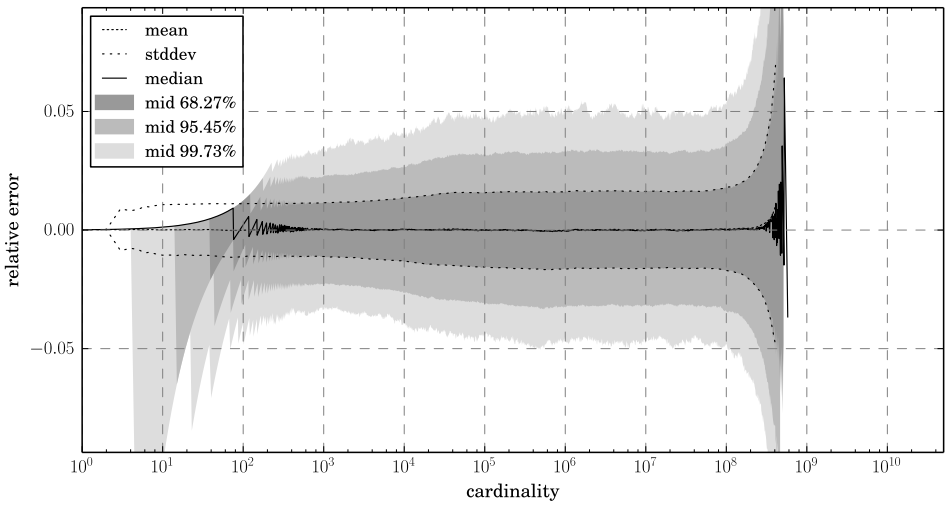

Figure 10: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 14$.

executions for each cardinality value. The results for HyperLogLog configurations $p = 12, q = 20$ and $p = 12, q = 52$ are shown in Figure 11. Two variants of Algorithm 3 have been evaluated for which the functions σ and τ have been either calculated on-demand or taken from a lookup table. All these benchmarks where carried out on an Intel Core i5-2500K clocking at 3.3 GHz.

The results show that the execution times are nearly constant for $q = 52$. Using a lookup table makes not much difference, because the on-demand calculation for σ is very fast and the calculation of τ is rarely needed due to the small probability of saturated registers ($c_{53} > 0$) for realistic cardinalities.

For the case $q = 20$ with precalculated correction values the computation time is again – as expected – independent of the cardinality. The faster computation times compared to the $q = 52$ case can be explained by the much smaller dimension of the multiplicity vector which is equal to $q + 2$. If the functions σ and τ are calculated on-demand, the execution times for larger cardinalities are doubled. This comes from the fact that the calculation of τ is much more expensive than that of σ, because it requires more iterations and involves square root evaluations. However, we can imagine that a better numerical approximation of τ can be found than that given in Algorithm 3 and which allows on-demand evaluation without significant extra costs.

The numbers do not yet include the required processing time to extract the multiplicity vector out from the HyperLogLog sketch, which requires a complete scan over all registers and counting the different register values into an array. A theoretical lower bound for this processing time can be derived using the maximum memory bandwidth of the CPU, which is 21 GB/s for an Intel Core i5-2500K. For precision $p = 12$ there are 4096 registers, each of them requires at least 5 bits. Hence, the total data size of the HyperLogLog sketch is 2.5 kB minimum and transfer time from main memory to CPU will be at least 120 ns. Having this value in mind, the presented numbers for estimating the cardinality from the multiplicity vector are quite satisfying.

4. Maximum likelihood estimation

Moreover, we know from Section 2.4 that any unbiased estimator for the Poisson parameter is also an unbiased estimator for the cardinality. We know that under suitable regularity conditions of the probability mass function the maximum likelihood estimator is asymptotically efficient [15]. This means, if the number of registers m is large enough, the maximum likelihood method should give us an unbiased estimator for the cardinality.

We are not the first applying the maximum likelihood method to HyperLogLog sketches [16]. However, there a different element insertion algorithm was considered, which is very inefficient in practice due to its $\mathcal{O}(m)$ runtime complexity. Each register has its own hash function which is applied to the inserted element. The result is used to update the corresponding register. The advantage is that the register values are statistically independent in this case, which allows factorization of the joint probability distribution of all register values.

In contrast, Algorithm 1 uses what is also called stochastic averaging. Instead of

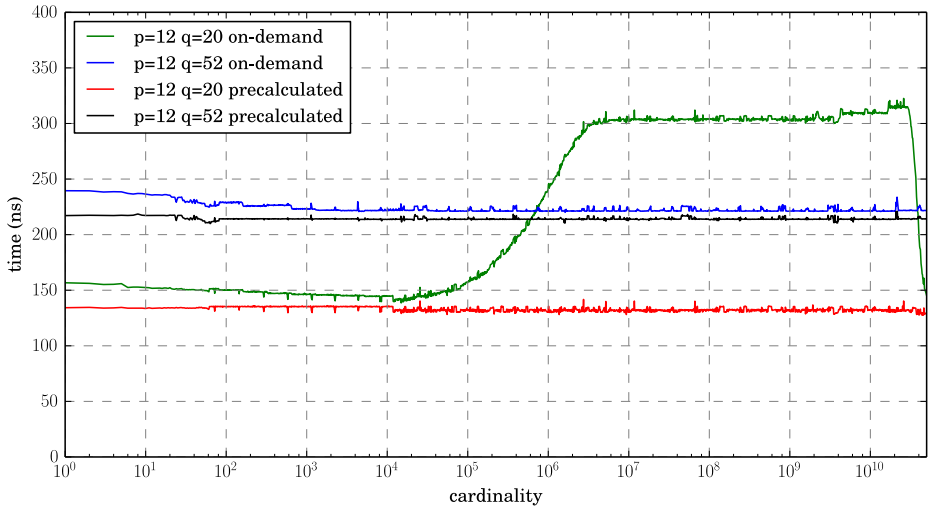

Figure 11: Average computation time as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3 GHz when estimating the cardinality from HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively. Both cases, σ and τ precalculated and calculated on-demand have been considered.

iterating through all registers, a single hash value is used, to select a single register and to update its value. Since the register values are statistically dependent in this case, the maximum likelihood method is much more complicated to apply. However, with the help of the Poisson approximation we are finally able to derive a new robust and efficient cardinality estimation algorithm. Furthermore, we will demonstrate that consequent application of the maximum likelihood method reveals that the cardinality estimate needs to be roughly proportional to the harmonic mean for intermediate cardinality values. The history of the HyperLogLog algorithm shows that the raw estimate (2) was first found after several attempts using the geometric mean [6, 14].

4.1. Log-likelihood function

Under the Poisson model the log-likelihood and its derivative are given by

$$\log \mathcal{L}(\lambda|\vec{k}) = -\frac{\lambda}{m}\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \log\left(1 - e^{-\frac{\lambda}{m2^k}}\right) + c_{q+1}\log\left(1 - e^{-\frac{\lambda}{m2^q}}\right) \tag{34}$$

and

$$\frac{d}{d\lambda}\log \mathcal{L}(\lambda|\vec{k}) = -\frac{1}{\lambda}\left(\frac{\lambda}{m}\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \frac{\frac{\lambda}{m2^k}}{1 - e^{\frac{\lambda}{m2^k}}} + c_{q+1}\frac{\frac{\lambda}{m2^q}}{1 - e^{\frac{\lambda}{m2^q}}}\right). \tag{35}$$

As a consequence, the maximum likelihood estimate for the Poisson parameter is given by

$$\hat{\lambda} = m\hat{x}, \tag{36}$$

if $\hat{x}$ is the root of the function

$$f(x) := x\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \frac{\frac{x}{2^k}}{1 - e^{\frac{x}{2^k}}} + c_{q+1}\frac{\frac{x}{2^q}}{1 - e^{\frac{x}{2^q}}}. \tag{37}$$

This function can also be written as

$$f(x) := x\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{x}{2^k}\right) + c_{q+1}h\left(\frac{x}{2^q}\right) - (m - c_0), \tag{38}$$

where the function $h(x)$ is defined as

$$h(x) := 1 - \frac{x}{e^x - 1}. \tag{39}$$

$h(x)$ is strictly increasing and concave as can be seen in Figure 12. For non-negative values x the function ranges from $h(0) = 0$ to $h(x \to \infty) = 1$. Since the function $f(x)$ is also strictly increasing, it is obvious that there exists a unique root $\hat{x}$ for which $f(\hat{x}) = 0$. The function is non-positive at 0 since $f(0) = c_0 - m \leq 0$ and, in case $c_{q+1} < m$ which implies $\sum_{k=0}^{q}\frac{c_k}{2^k} > 0$, the function is at least linearly increasing. $c_{q+1} = m$ corresponds to the case with all registers equal to the maximum value $(q+1)$, for which the maximum likelihood estimate would be infinity.

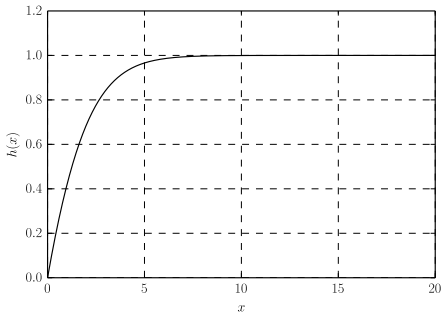

Figure 12: The function $h(x)$.

It is easy to see that the estimate $\hat{\lambda}$ remains equal or becomes larger, when inserting an element into the HyperLogLog sketch following Algorithm 1. An update potentially changes the multiplicity vector $(c_0, \ldots, c_{q+1})$ to $(c_0, \ldots, c_i - 1, \ldots, c_j + 1, \ldots, c_{q+1})$ where $i < j$. Writing (38) as

$$f(x) := c_0 x + c_1 \left(h\Big(\frac{x}{2^1}\Big) + \frac{x}{2^1} - 1 \right) + c_2 \left(h\Big(\frac{x}{2^2}\Big) + \frac{x}{2^2} - 1 \right) + \ldots \\ \cdots + c_q \left(h\Big(\frac{x}{2^q}\Big) + \frac{x}{2^q} - 1 \right) + c_{q+1} \left(h\Big(\frac{x}{2^q}\Big) - 1 \right). \quad (40)$$

shows that the coefficient of c_i is larger than the coefficient of c_j in case $i < j$. Keeping x fixed during an update decreases $f(x)$. As a consequence, since $f(x)$ is increasing, the new root and hence the estimate must be larger than before the update.

Note that (37) can be solved analytically for the special case $q = 0$ which corresponds to the already mentioned linear counting algorithm. In this case, the maximum likelihood method under the Poisson model directly leads to the linear counting estimator presented in [12] and which was also used for small range estimation (14). Despite the assumption of a Poisson model, it is a very good approximation of the optimal martingale estimator presented in [3]. Due to this fact we could expect that maximum likelihood estimation under the Poisson model also works well for the more general HyperLogLog case.

4.2. Inequalities for the maximum likelihood estimate

In the following lower and upper bounds for $\hat{x}$ are derived. Applying Jensen's inequality on h in (38) gives an upper bound for $f(x)$:

$$f(x) \leq x \sum_{k=0}^{q} \frac{c_k}{2^k} + (m - c_0) \cdot h\left(x \cdot \left(\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q} \right) \right) - (m - c_0). \quad (41)$$

The left-hand side is zero, if $\hat{x}$ is inserted. Resolution for $\hat{x}$ finally gives the lower bound

$$\hat{x} \geq \frac{m - c_0}{\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}} \log\left(1 + \frac{\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}{\sum_{k=0}^{q} \frac{c_k}{2^k}} \right). \quad (42)$$

This bound can be weakened using $\log(1+x) \geq \frac{2x}{x+2}$ for $x \geq 0$ which results in

$$\hat{x} \geq \frac{m - c_0}{c_0 + \frac{3}{2}\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}. \tag{43}$$

Using the monotonicity of h, the lower bound

$$f(x) \geq x \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\Big(\frac{x}{2^{k'_{\max}}}\Big) + c_{q+1} h\Big(\frac{x}{2^{k'_{\max}}}\Big) - (m - c_0)\,, \tag{44}$$

for f can be found, where $k'_{\max} := \min(k_{\max}, q)$ and $k_{\max} := \max\{k | c_k > 0\}$. Again, inserting $\hat{x}$ and transformation gives

$$\hat{x} \leq 2^{k'_{\max}} \log\left(1 + \frac{m - c_0}{2^{k'_{\max}} \sum_{k=0}^{q} \frac{c_k}{2^k}}\right) \tag{45}$$

as upper bound which can be weakened using $\log(1+x) \leq x$ for $x \geq 0$

$$\hat{x} \leq \frac{m - c_0}{\sum_{k=0}^{q} \frac{c_k}{2^k}}. \tag{46}$$

If the HyperLogLog sketch is in the intermediate range, where $c_0 = c_{q+1} = 0$ the bounds (43) and (46) differ only by a constant factor from the raw estimator (11). Hence, the maximum likelihood method leads directly to the harmonic that is used by the raw estimator.

4.3. Computation of the maximum likelihood estimate

Since f is concave and increasing, both, Newton-Raphson iteration and the secant method, converge to the root, if the function is negative for the starting points. In the following we start from the secant method to derive the new cardinality estimation algorithm. Even though the secant method has the disadvantage of slower convergence, a single iteration is simpler to calculate as it does not require the evaluation of the first derivative.

An iteration step of the secant method can be written as

$$x_i = x_{i-1} - (x_{i-1} - x_{i-2}) \frac{f(x_{i-1})}{f(x_{i-1}) - f(x_{i-2})} \tag{47}$$

If $x_0 = 0$ with $f(x_0) = -(m - c_0)$, and x_1 is equal to one of the derived lower bounds (42) or (43), the sequence $\{x_i\}$ is montone increasing. Using the definitions

$$\Delta x_i := x_i - x_{i-1} \tag{48}$$

and

$$g(x) := f(x) + (m - c_0) = x \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\Big(\frac{x}{2^k}\Big) + c_{q+1} h\Big(\frac{x}{2^q}\Big) \tag{49}$$

the iteration scheme can also be written as

$$\Delta x_i = \Delta x_{i-1} \frac{(m - c_0) - g(x_{i-1})}{g(x_{i-1}) - g(x_{i-2})} \tag{50}$$

$$x_i = x_{i-1} + \Delta x_i \tag{51}$$

The iteration can be stopped, if $\Delta x_i \leq \delta \cdot x_i$. Since the expected statistical errror for the HyperLogLog data structure scales according to $\frac{1}{\sqrt{m}}$ [6], it makes sense to choose $\delta = \frac{\varepsilon}{\sqrt{m}}$ with some constant ε. For all results that will follow in Section 4.5 we have used $\varepsilon = 10^{-2}$.

4.4. Maximum likelihood estimation algorithm

In order to get a fast cardinality estimation algorithm, it is crucial to minimize the evaluation costs for (49). A couple of optimizations allow significant reduction of the computational effort:

- Only a fraction of all count values c_k is non-zero. If we denote $k_{\min} := \min\{k|c_k > 0\}$ and $k_{\max} := \max\{k|c_k > 0\}$, it is sufficient to loop over all indices in the range $[k_{\min}, k_{\max}]$.
- The coefficient of the linear term $\sum_{k=0}^{q} \frac{c_k}{2^k}$ can be precalculated and reused for all function evaluations.
- Many programming languages allow the efficient multiplication and division by any integral power of 2 using special functions, such as `ldexp` in C/C++ or `scalb` in Java.
- The function $h(x)$ only needs to be evaluated at values $\left\{ \frac{x}{2^{k'_{\max}}}, \frac{x}{2^{k'_{\max}-1}} \cdots, \frac{x}{2^{k_{\min}}} \right\}$ where $k'_{\max} := \min(k_{\max}, q)$. This series corresponds to a geometric series with ratio 2. A straightforward calculation using (39) is very expensive because of the exponential function. However, if we already know $h\left(\frac{x}{2^{k'_{\max}}}\right)$ all other required function values can be easily obtained using the identity

  $$h(2x) = \frac{x + 2h(x)\,(1 - h(x))}{x + 2\,(1 - h(x))} \tag{52}$$

  or

  $$h\Big(\frac{x}{2^k}\Big) = \frac{\frac{x}{2^{k+2}} + h\big(\frac{x}{2^{k+1}}\big)\,\big(1 - h\big(\frac{x}{2^{k+1}}\big)\big)}{\frac{x}{2^{k+2}} + \big(1 - h\big(\frac{x}{2^{k+1}}\big)\big)} \tag{53}$$

  Note, this recursive formula is stable in a sense that the relative error of $h(2x)$ is smaller than that of $h(x)$ as shown in Appendix A.
- If $h\left(\frac{x}{2^{k'_{\max}}}\right)$ is smaller than 0.5, the function $h(x)$ can be well approximated by a Taylor series around $x = 0$

  $$h(x) = \frac{x}{2} - \frac{x^2}{12} + \frac{x^4}{720} - \frac{x^6}{30240} + \mathcal{O}(x^8) \tag{54}$$

which can be optimized for numerical evaluation using Estrin's scheme and $x' := \frac{x}{2}$ and $x'' := x'x'$

$$h(x) = x' - x''/3 + \left(x''x''\right)\left(1/45 - x''/472.5\right) + \mathcal{O}(x^8) \tag{55}$$

In fact, $h\left(\frac{x}{2^{k'_{\max}}}\right) \leq 0.5$ is almost always fulfilled as long as registers are not saturated. Using (45) it is straightforward to see that $\frac{x}{2^{k'_{\max}}} \leq \log 2 \approx 0.693$, if $c_{q+1} = 0$. In case $\frac{x}{2^{k'_{\max}}} > 0.5$, the value $\frac{x}{2^{e+1}}$ is taken instead, where e is the exponent of the floating point representation of x, $e = 1 + \lfloor \log_2(x) \rfloor$. By definition, $\frac{x}{2^{e+1}} \leq 0.5$ which allows using the Taylor series approximation. $h\left(\frac{x}{2^{k'_{\max}}}\right)$ is finally obtained after $e + 1 - k'_{\max}$ iterations using (53).

All these optimizations together finally give the new cardinality estimation algorithm presented in Algorithm 4.

4.5. Estimation error

In order to investigate the estimation error for the maximum likelihood estimation algorithm, we investigated the same HyperLogLog configurations as in Section 3.5 for the corrected raw estimation algorithm. Figure 13, Figure 14, Figure 15,Figure 16,Figure 17, and Figure 18 show very similar results for various HyperLogLog parameters.

What is different for the maximum likelihood estimation approach is a somewhat smaller median bias with less oscillations around zero for small cardinalities. Furthermore, contrary to the raw estimator which reveals a small oscillating bias for the mean (see Figure 8), the maximum likelihood estimator seems to be completely unbiased (see Figure 16).

4.6. Performance

We also measured the performance of Algorithm 4 using the same test setup as described in Section 3.6. The results for HyperLogLog configurations $p = 12, q = 20$ and $p = 12, q = 52$ are shown in Figure 19. The average computation time for the maximum likelihood algorithm shows a different behavior than for the corrected raw estimation algorithm (compare Figure 11). As can be seen, the average execution time is larger for most cardinalities, but nevertheless it never exceeds 700 ns which is still fast enough for many applications.

5. Cardinality estimation of set intersections and differences

Assume two different sets S_1 and S_2 that have been recorded by two HyperLogLog sketches with same parameters (p, q). Given the corresponding register values $\vec{k}_1$ and $\vec{k}_2$, we attempt to estimate the cardinalities of the pair-wise disjoint sets $X = S_1 \cap S_2$, $A = S_1 \setminus S_2$, and $B = S_2 \setminus S_1$. Motivated by the good results we have obtained for cardinality estimation of a single HyperLogLog sketch, we want to get these estimates

Algorithm 4 Maximum likelihood cardinality estimation.

function EstimateCardinality($\vec{c}$)
  $q \leftarrow \dim(\vec{c}) - 2$
  $k_{\min} \leftarrow \min\{k | c_k > 0\}$
  if $k_{\min} > q$ **then**
    return ∞
  end if
  $k'_{\min} \leftarrow \max(k_{\min}, 1)$
  $k_{\max} \leftarrow \max\{k | c_k > 0\}$
  $k'_{\max} \leftarrow \min(k_{\max}, q)$
  $z \leftarrow 0$
  $m' \leftarrow c_{q+1}$
  $y \leftarrow 2^{-k'_{\max}}$
  for $k \leftarrow k'_{\max}, k'_{\min}$ **do**
    $z \leftarrow z + c_k \cdot y$ ▷ here $y = 2^{-k}$
    $y \leftarrow 2y$
    $m' \leftarrow m' + c_k$
  end for
  $m \leftarrow m' + c_0$
  $c' \leftarrow c_{q+1}$
  if $q \geq 1$ **then**
    $c' \leftarrow c' + c_{k'_{\max}}$
  end if
  $g_{\text{prev}} \leftarrow 0$
  $a \leftarrow z + c_0$
  $b \leftarrow z + c_{q+1} \cdot 2^{-q}$
  if $b \leq 1.5 \cdot a$ **then**
    $x \leftarrow m'/(0.5 \cdot b + a)$ ▷ weak lower bound (43)
  else
    $x \leftarrow m'/b \cdot \log(1 + b/a)$ ▷ strong lower bound (42)
  end if

Algorithm 4 Maximum likelihood cardinality estimation (continued).

$\Delta x \leftarrow x$
while $\Delta x > x \cdot \varepsilon$ **do** ▷ secant method iteration, $\varepsilon = 10^{-2}$
  $e \leftarrow 1 + \lfloor \log_2(x) \rfloor$
  $x' \leftarrow x \cdot 2^{-\max(k'_{\max}+1, e+2)}$ ▷ $x' \in [0, 0.25]$
  $x'' \leftarrow x' \cdot x'$
  $h \leftarrow x' - x''/3 + (x'' \cdot x'') \cdot (1/45 - x''/472.5)$ ▷ Taylor approximation (55)
  for $k \leftarrow e, k'_{\max}$ **do**
    $h \leftarrow \frac{x'+h\cdot(1-h)}{x'+(1-h)}$ ▷ calculate $h\left(\frac{x}{2^k}\right)$, see (53), at this point $x' = \frac{x}{2^{k+2}}$
    $x' \leftarrow 2x'$
  end for
  $g \leftarrow c' \cdot h$ ▷ compare (49)
  for $k \leftarrow (k'_{\max} - 1), k'_{\min}$ **do**
    $h \leftarrow \frac{x'+h\cdot(1-h)}{x'+(1-h)}$ ▷ calculate $h\left(\frac{x}{2^k}\right)$, see (53), at this point $x' = \frac{x}{2^{k+2}}$
    $g \leftarrow g + c_k \cdot h$
    $x' \leftarrow 2x'$
  end for
  $g \leftarrow g + x \cdot a$
  if $g > g_{\text{prev}} \wedge m' \geq g$ **then**
    $\Delta x \leftarrow \Delta x \cdot \frac{m'-g}{g-g_{\text{prev}}}$ ▷ see (50)
  else
    $\Delta x \leftarrow 0$
  end if
  $x \leftarrow x + \Delta x$
  $g_{\text{prev}} \leftarrow g$
end while
return $m \cdot x$
end function

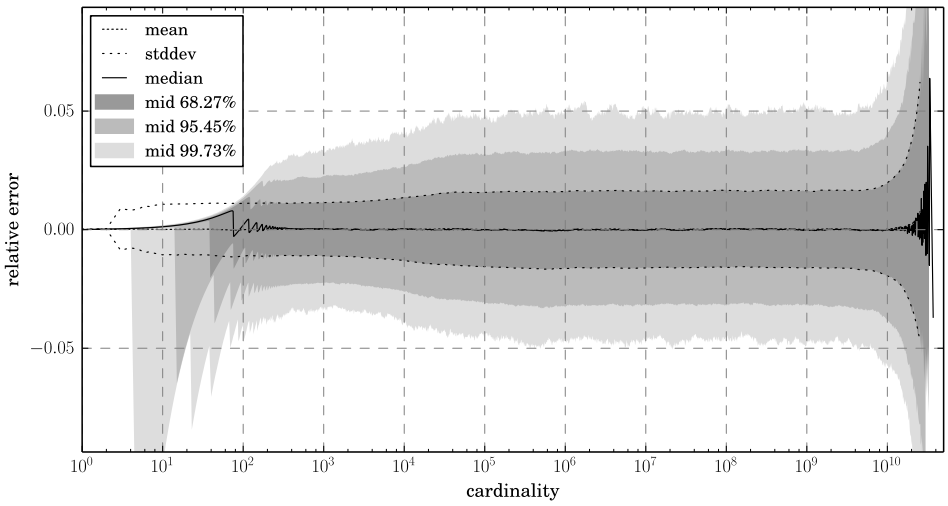

Figure 13: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

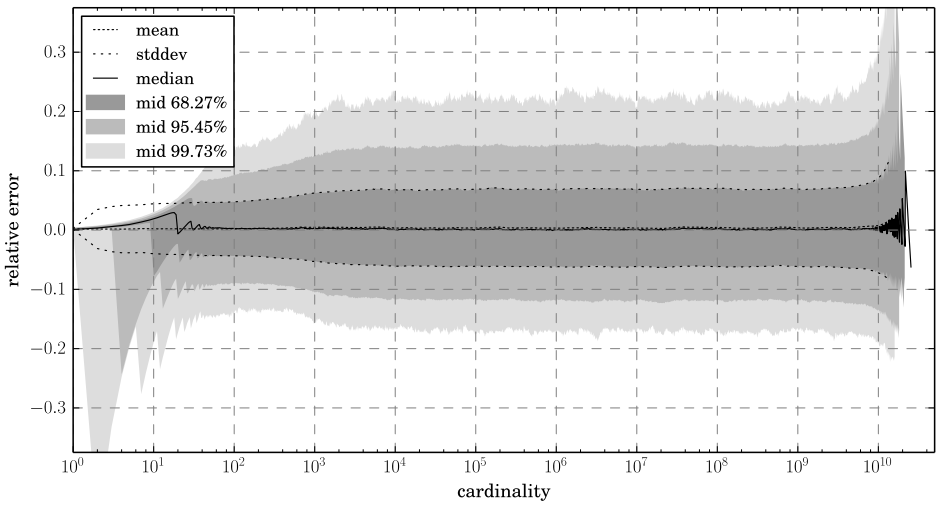

Figure 14: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

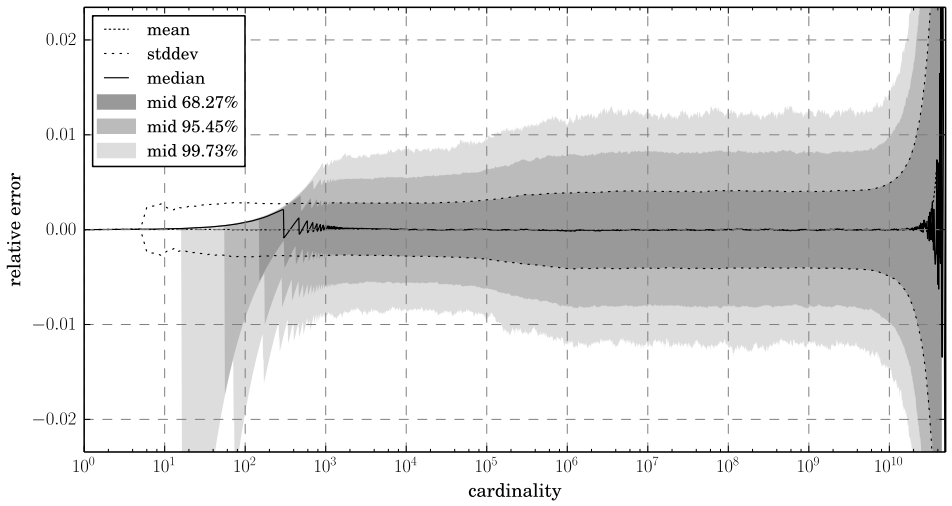

Figure 15: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

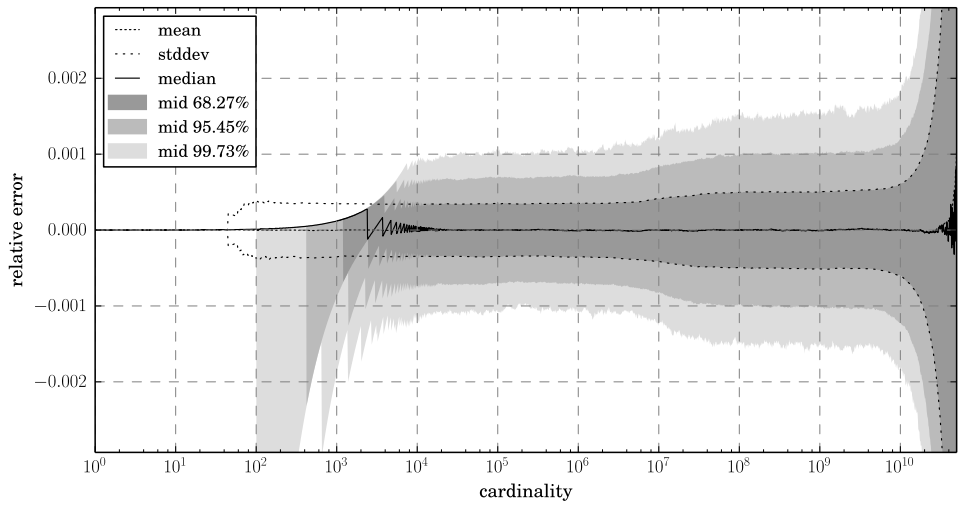

Figure 16: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 10$.

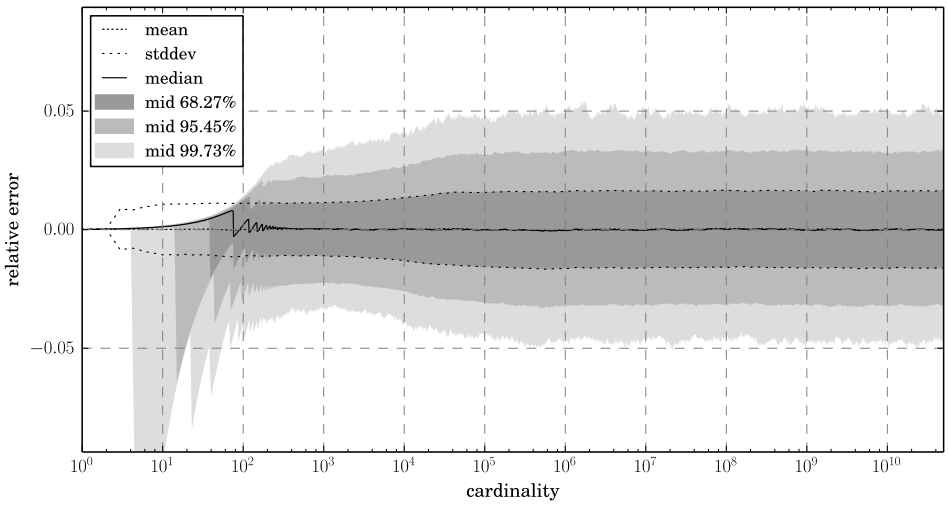

Figure 17: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 52$.

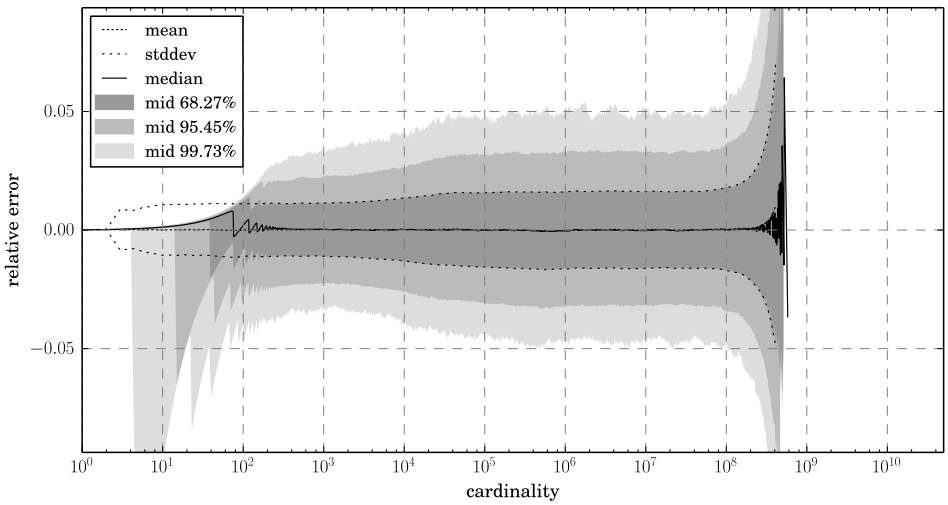

Figure 18: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 14$.

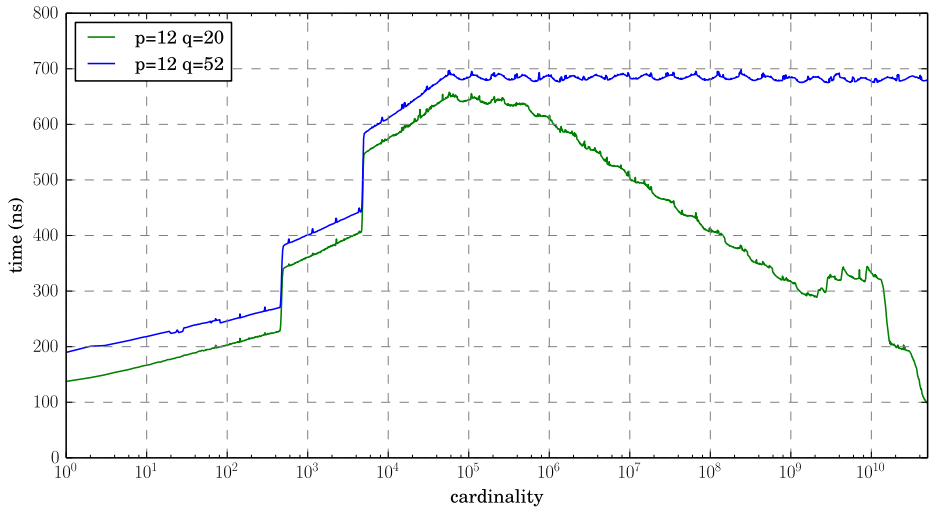

Figure 19: Average execution time of the maximum likelihood estimation algorithm as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3 GHz for HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively.

using the maximum likelihood method applied to the joint probability distribution of $\vec{k}_1$ and $\vec{k}_2$.

Under the Poisson model the register values are independent and identically distributed. Therefore, we first derive the joint probability distribution for a single register that has value K_1 in the first HyperLogLog sketch representing S_1 and value K_2 in the second HyperLogLog sketch representing S_2.

The HyperLogLog sketch that represents S_1 could have also been obtained by constructing two HyperLogLog sketches from sets A and X and by merging both by taking for each register the maximum value of both sketches. Analogously, the HyperLogLog sketch for S_2 could have been obtained from sketches for B and X. Let us consider the register values K_a, K_b, and K_x at a certain position of the HyperLogLog sketches for A, B, and X, respectively. The corresponding values in sketches for S_1 and S_2 are given by

$$K_1 = \max(K_a, K_x)\,, \quad K_2 = \max(K_b, K_x) \tag{56}$$

Their joint cumulative probability function is given as

$$\begin{aligned} P(K_1 \leq k_1 \wedge K_2 \leq k_2) &= P(\max(K_a, K_x) \leq k_1 \wedge \max(K_b, K_x) \leq k_2) \\ &= P(K_a \leq k_1 \wedge K_b \leq k_2 \wedge K_x \leq \min(k_1, k_2)) \\ &= P(K_a \leq k_1)\, P(K_b \leq k_2)\, P(K_x \leq \min(k_1, k_2)) \end{aligned} \tag{57}$$

Here the last transformation used the independence of K_a, K_b, and K_x, because by definition, the sets A, B, and X are disjoint. Furthermore, under the Poisson model K_a,

K_b, and K_x obey (6). If we assume that elements are added to A, B, and X at rates λ_x, λ_a, and λ_b, respectively, the probability that a certain register has a value less than or equal to k_1 in the first HyperLogLog sketch and simultaneously a value less than or equal to k_2 in the second one can be written as

$$P(K_1 \leq k_1 \wedge K_2 \leq k_2) = \begin{cases} 0 & k_1 < 0 \vee k_2 < 0 \\ e^{-\frac{\lambda_a}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}} - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}} & 0 \leq k_1 \leq q \wedge 0 \leq k_2 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}} & 0 \leq k_2 \leq q < k_1 \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}} & 0 \leq k_1 \leq q < k_2 \\ 1 & q < k_1 = k_2 \end{cases} \tag{58}$$

The joint probability mass function for both register values can be calculated using

$$\begin{aligned} \rho(k_1, k_2) = {} & P(K_1 \leq k_1 \wedge K_2 \leq k_2) - P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2) \\ & - P(K_1 \leq k_1 \wedge K_2 \leq k_2 - 1) + P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2 - 1) \end{aligned} \tag{59}$$

which finally gives

$$\rho(k_1, k_2) = \begin{cases} e^{-\frac{\lambda_a+\lambda_x}{m} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 0 = k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_b}{m2^q}}\right) & 0 = k_1 < k_2 = q + 1 \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right) \left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 1 \leq k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right) \left(1 - e^{-\frac{\lambda_b}{m2^q}}\right) & 1 \leq k_1 < k_2 = q + 1 \\ e^{-\frac{\lambda_b+\lambda_x}{m} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 0 = k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_a}{m2^q}}\right) & 0 = k_2 < k_1 = q + 1 \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right) \left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 1 \leq k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right) \left(1 - e^{-\frac{\lambda_a}{m2^q}}\right) & 1 \leq k_2 < k_1 = q + 1 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m}} & 0 = k_1 = k_2 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^k}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^k}} - e^{-\frac{\lambda_b+\lambda_x}{m2^k}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^k}}\right) & 1 \leq k_1 = k_2 = k \leq q \\ 1 - e^{-\frac{\lambda_a+\lambda_x}{m2^q}} - e^{-\frac{\lambda_b+\lambda_x}{m2^q}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^q}} & k_1 = k_2 = q + 1 \end{cases} \tag{60}$$

The logarithm of the joint probability mass function can be written using Iverson

bracket notation ([true] := 1, [false] := 0) as

$$\begin{aligned}\log(\rho(k_1,k_2)) = &-\frac{\lambda_a}{m2^{k_1}}\,[k_1 \le q] - \frac{\lambda_b}{m2^{k_2}}\,[k_2 \le q] - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}\,[k_1 \le q \vee k_2 \le q] \\ &+ \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right)[1 \le k_1 < k_2] \\ &+ \log\left(1 - e^{-\frac{\lambda_a}{m2^{\min(k_1,q)}}}\right)[k_2 < k_1] \\ &+ \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right)[1 \le k_2 < k_1] \\ &+ \log\left(1 - e^{-\frac{\lambda_b}{m2^{\min(k_2,q)}}}\right)[k_1 < k_2] \\ &+ \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k_1,q)}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}}\right)[1 \le k_1 = k_2]\end{aligned} \tag{61}$$

Since under the Poisson model, the values for different registers are independent we are now able to write the joint probability mass function for the joint state of both HyperLogLog sketches

$$\rho(\vec{k}_1, \vec{k}_2) = \prod_{k_1=0}^{q+1}\prod_{k_2=0}^{q+1} \rho(k_1,k_2)^{c_{k_1k_2}}. \tag{62}$$

Here we have used the multiplicity matrix $\mathbf{c} = (c_{k_1k_2})_{k_1k_2 \in [0,q+1]}$ defined by

$$c_{k_1k_2} := |\{(i,j)\,|k_{i,1} = k_1 \wedge k_{j,2} = k_2\}| \tag{63}$$

As in Section 3 where we found that the multiplicity vector $\vec{c}$ of a sketch is a sufficient statistic for the cardinality, the multiplicity matrix of two sketches is a sufficient statistic for the cardinalities of X, A, and B.

In order to get the maximum likelihood estimates $\hat{\lambda}_a$, $\hat{\lambda}_b$, and $\hat{\lambda}_x$ we need to maximize the log-likelihood function given by

$$\log \mathcal{L}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = \sum_{k_1=0}^{q+1}\sum_{k_2=0}^{q+1} c_{k_1k_2} \log(\rho(k_1,k_2)) \tag{64}$$

Insertion of (61) results in

$$\begin{aligned}\log \mathcal{L}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{\lambda_a}{m} \sum_{k=0}^{q} \frac{c_k^{\leftarrow} + c_{kk} + c_k^{\rightarrow}}{2^k} - \frac{\lambda_b}{m} \sum_{k=0}^{q} \frac{c_k^{\uparrow} + c_{kk} + c_k^{\downarrow}}{2^k} \\ &- \frac{\lambda_x}{m} \sum_{k=0}^{q} \frac{c_k^{\rightarrow} + c_{kk} + c_k^{\downarrow}}{2^k} \\ &+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k,q)}}}\right) c_k^{\rightarrow} + \log\left(1 - e^{-\frac{\lambda_a}{m2^{\min(k,q)}}}\right) c_k^{\leftarrow} \\ &+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k,q)}}}\right) c_k^{\downarrow} + \log\left(1 - e^{-\frac{\lambda_b}{m2^{\min(k,q)}}}\right) c_k^{\uparrow} \\ &+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k,q)}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k,q)}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{\min(k,q)}}}\right) c_{kk}\end{aligned} \tag{65}$$

which is the two HyperLogLog case analog of (34). The constants $c_k^{\uparrow} := \sum_{i=0}^{k-1} c_{ik}$, $c_k^{\rightarrow} := \sum_{i=k+1}^{q+1} c_{ki}$, $c_k^{\downarrow} := \sum_{i=k+1}^{q+1} c_{ik}$, and $c_k^{\leftarrow} := \sum_{i=0}^{k-1} c_{ki}$ correspond to sums within the multiplicity matrix, which are obtained by aggregating all elements that are in up, right, down, and left directions relative to the k-th diagonal entry c_{kk}, respectively.

The log-likelihood function (65) does not always have a strict global maximum point. For example, in case $\mathbf{c}$ is a strict lower triangular matrix which corresponds to the case that each register of the first HyperLogLog sketch is larger than the corresponding value in the second HyperLogLog sketch, the function can be rewritten as sum of two functions, one dependent on λ_a and the other dependent on $(\lambda_b + \lambda_x)$. The maximum is obtained, if $\lambda_a = \hat{\lambda}_1$ and $\lambda_b + \lambda_x = \hat{\lambda}_2$. Here $\hat{\lambda}_1$ and $\hat{\lambda}_2$ are the cardinality estimates for the first and second HyperLogLog sketch, respectively. Similarly, if all register values of the first are smaller than those of the second HyperLogLog sketch, the maximum is obtained when $\lambda_a + \lambda_x = \hat{\lambda}_1$ and $\lambda_b = \hat{\lambda}_2$.

If we know that the two HyperLogLog sketches have been filled by elements taken from disjoint sets we can assume $\lambda_x = 0$. In this case (65) is separable into the sum of two log-likelihood functions that depend on λ_a and λ_b, respectively, and that follow (34). Hence, the joint maximum likelihood estimation of λ_a and λ_b gives the same results as estimating them independently by maximizing (34).

The first derivatives are

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \lambda_a}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{1}{m} \sum_{k=0}^{q} \frac{c_k^{\leftarrow} + c_{kk} + c_k^{\rightarrow}}{2^k} + \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a+\lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\rightarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\leftarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - 1}{e^{\frac{\lambda_a+\lambda_b+\lambda_x}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + 1} \frac{c_{kk}}{2^{\min(k,q)}}\end{aligned} \tag{66}$$

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \lambda_x}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{1}{m} \sum_{k=0}^{q} \frac{c_k^{\rightarrow} + c_{kk} + c_k^{\downarrow}}{2^k} + \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a+\lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\rightarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_b+\lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\downarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - 1}{e^{\frac{\lambda_a+\lambda_b+\lambda_x}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + 1} \frac{c_{kk}}{2^{\min(k,q)}}\end{aligned} \tag{67}$$

5.1. Results

TODO

6. Conclusion and future work

TODO

A. Numerical stability of recursion formula for $h(x)$

In order to investigate the error propagation of a single recursion step using (52) we define $h_1 := h(x)$ and $h_2 := h(2x)$. The recursion formula simplifies to

$$h_2 = \frac{x + 2h_1(1 - h_1)}{x + 2(1 - h_1)}. \tag{68}$$

If h_1 is approximated by $\tilde{h}_1 = h_1 (1 + \varepsilon_1)$ with relative error ε_1, the recursion formula will give an approximation for h_2

$$\tilde{h}_2 = \frac{x + 2\tilde{h}_1(1 - \tilde{h}_1)}{x + 2(1 - \tilde{h}_1)} \tag{69}$$

The corresponding relative error ε_2 is given by

$$\varepsilon_2 = \frac{\tilde{h}_2}{h_2} - 1. \tag{70}$$

Putting (68) and (69) into (70) and using the first-order approximations

$$\frac{x + 2\tilde{h}_1(1-\tilde{h}_1)}{x + 2h_1(1-h_1)} = 1 + \varepsilon_1 \frac{2h_1\,(1-2h_1)}{x + 2h_1\,(1-h_1)} + \mathcal{O}(\varepsilon_1^2) \tag{71}$$

and

$$\frac{x + 2(1-h_1)}{x + 2(1-\tilde{h}_1)} = 1 + \varepsilon_1 \frac{2h_1}{x + 2\,(1-h_1)} + \mathcal{O}(\varepsilon_1^2), \tag{72}$$

we obtain

$$\varepsilon_2 = \varepsilon_1 \left(\frac{2h_1\,(1-2h_1)}{x + 2h_1\,(1-h_1)} + \frac{2h_1}{x + 2\,(1-h_1)} \right) + \mathcal{O}(\varepsilon_1^2) \tag{73}$$

By numerical means it is easy to show that

$$0 \leq \frac{2h(x)\,(1-2h(x))}{x + 2h(x)\,(1-h(x))} + \frac{2h(x)}{x + 2\,(1-h(x))} \leq 0.517 \tag{74}$$

holds for all $x \geq 0$. Therefore,

$$|\varepsilon_2| \leq 0.517\ |\varepsilon_1| + \mathcal{O}(\varepsilon_1^2) \tag{75}$$

which means that the relative error is decreasing in each recursion step and the recursive calculation of h is numerically stable.

B. Error caused by approximation of $h(x)$

According to (38) the exact estimate $\hat{x}$ fulfills

$$\hat{x} \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{\hat{x}}{2^k}\right) + c_{q+1} h\left(\frac{\hat{x}}{2^q}\right) = m - c_0 \tag{76}$$

If h is not calculated exactly but approximated by $\tilde{h}$ with maximum relative error $\varepsilon_h \ll 1$

$$\left|\tilde{h}(x) - h(x)\right| \leq \varepsilon_h h(x) \tag{77}$$

the solution of the equation will be off by a relative error ε_x:

$$\hat{x}\,(1+\varepsilon_x) \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right) + c_{q+1} \tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right) = m - c_0 \tag{78}$$

Due to (77) there exists some $\alpha \in [-\varepsilon_h, \varepsilon_h]$ for which

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{c_k}{2^k} + (1+\alpha)\left(\sum_{k=1}^{q} c_k h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right) + c_{q+1}h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right)\right) = m - c_0 \tag{79}$$

Since $h'(x) \in [0, 0.5]$ for $x \geq 0$, there exists a $\beta \in [0, 0.5]$ for which

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{c_k}{2^k} + (1+\alpha)\left(\sum_{k=1}^{q} c_k h\left(\frac{\hat{x}}{2^k}\right) + \frac{c_k}{2^k}\hat{x}\varepsilon_x\beta\right) + \\ + (1+\alpha)\left(c_{q+1}h\left(\frac{\hat{x}}{2^q}\right) + \frac{c_{q+1}}{2^q}\hat{x}\varepsilon_x\beta\right) = m - c_0 \tag{80}$$

Subtracting (76) multiplied by $(1+\alpha)$ from (80) and resolving ε_x gives

$$\varepsilon_x = \alpha\frac{\hat{x}\sum_{k=0}^{q}\frac{c_k}{2^k} - (m-c_0)}{\hat{x}\left(\sum_{k=0}^{q}\frac{c_k}{2^k} + (1+\alpha)\,\beta\left(\sum_{k=1}^{q}\frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}\right)\right)} \tag{81}$$

Using $|\alpha| \leq \varepsilon_h$, $\beta \geq 0$, and (46) the absolute value of the relative error can be bounded by

$$|\varepsilon_x| \leq |\varepsilon_h|\,\frac{(m-c_0) - \hat{x}\sum_{k=0}^{q}\frac{c_k}{2^k}}{\hat{x}\sum_{k=0}^{q}\frac{c_k}{2^k}} \tag{82}$$

Furthermore, using (43) we finally get

$$|\varepsilon_x| \leq |\varepsilon_h|\,\frac{\frac{1}{2}\sum_{k=1}^{q}\frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}{\sum_{k=0}^{q}\frac{c_k}{2^k}} \leq |\varepsilon_h|\left(\frac{1}{2} + \frac{\frac{c_{q+1}}{2^q}}{\sum_{k=1}^{q}\frac{c_k}{2^k}}\right) \leq |\varepsilon_h|\left(\frac{1}{2} + \frac{c_{q+1}}{m - c_{q+1}}\right) \tag{83}$$

Hence, as long as most registers are not in the saturated state ($c_{q+1} \ll m$), the relative error ε_x of the calculated estimate using the approximation $\tilde{h}(x)$ for $h(x)$ has the same order of magnitude as ε_h.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137 – 147, 1999.

[2] Ahmed Metwally, Divyakant Agrawal, and Amr El 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*, pages 618–629, Nantes, France, March 2008.

[3] Daniel Ting. Streamed approximate counting of distinct elements: Beating optimal batch methods. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 442–451, New York, NY, USA, August 2014.

[4] P. Indyk and D. Woodruff. Tight lower bounds for the distinct elements problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 283–288, Cambridge, MA, USA, October 2003.

[5] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 41–52, Indianapolis, IN, USA, June 2010.

[6] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 13th Conference on Analysis of Algorithms*, pages 127–146, Juan des Pins, France, June 2007.

[7] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692, Genoa, Italy, March 2013.

[8] Lee Rhodes. System and method for enhanced accuracy cardinality estimation, September 24 2015. US Patent 20,150,269,178.

[9] Salvatore Sanfilippo. Redis new data structure: The HyperLogLog. `http://antirez.com/news/75`, 2014.

[10] Aiyou Chen, Jin Cao, Larry Shepp, and Tuan Nguyen. Distinct counting with a self-learning bitmap. *Journal of the American Statistical Association*, 106(495):879–890, 2011.

[11] Aiyou Chen, Jin Cao, and Lawrence E Menten. Adaptive distinct counting for network-traffic monitoring and other applications, January 6 2015. US Patent 8,931,088.

[12] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, June 1990.

[13] Philippe Jacquet and Wojciech Szpankowski. Analytical depoissonization and its applications. *Theoretical Computer Science*, 201(1):1–62, 1998.

[14] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *Proceedings of the 11th Annual European Symposium on Algorithms*, pages 605–617, Budapest, Hungary, September 2003.

[15] George Casella and Roger L Berger. *Statistical inference*. Duxbury, Pacific Grove, CA, USA, 2nd edition, 2002.

[16] Peter Clifford and Ioana A. Cosma. A statistical analysis of probabilistic counting algorithms. *Scandinavian Journal of Statistics*, 39(1):1–14, 2012.