New cardinality estimation algorithms for HyperLogLog sketches

Draft version

Otmar Ertl
otmar.ertl@gmail.com

August 7, 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

This paper together with source code for all presented algorithms and simulations is available at `https://github.com/oertl/hyperloglog-sketch-estimation-paper`.

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 zero initial value. 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 zeros. If the position of the first 1-bit exceeds the current 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.

The desribed element insertion algorithm makes use of what is known as stochastic averaging [12]. Instead of updating each of all m registers using m independent hash values, which would be an $\mathcal{O}(m)$ operation, only one register is selected and updated, which requires only a single hash function and reduces the complexity to $\mathcal{O}(1)$.

A HyperLogLog sketch can be characterized by a parameter pair (p, q). The first parameter p is the precision and 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 will be 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. First, the insertion order of elements has no influence on the final HyperLogLog sketch. Furthermore, any two HyperLogLog sketches with same parameters (p, q) representing sets A and B can be easily merged. The resulting register values are simply given by the register-wise maximum values

$$K_i^{A\cup B} = \max\left(K_i^A, K_i^B\right) \quad \text{for } 1 \le i \le m. \tag{1}$$

The final HyperLogLog sketch represents the union of A and B.

A $(p, 0)$-HyperLogLog sketch corresponds to a bit array as used by linear counting [13]. Each register value can be stored by a single bit in this case. 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 registers $\boldsymbol{K} = (K_1, \ldots, K_m)$ have zero initial value and remain in the range $[0, q + 1]$.

procedure InsertElement(D, $\boldsymbol{K}$)
  $\langle a_1, \ldots, a_p, b_1, \ldots, b_q\rangle_2 \leftarrow (p+q)$-bit hash value of D
  $i \leftarrow 1 + \langle a_1, \ldots, a_p\rangle_2$ ▷ $i \in [1, 2^p]$
  $k = \min(\{s \mid s \in [1, q] \wedge b_s = 1\} \cup \{q + 1\})$ ▷ $k \in [1, q + 1]$
  $K_i \leftarrow \max(K_i, k)$
end procedure

2.2. Joint probability distribution of register values

Under the assumption of a uniform hash function, the probability that the register values $\boldsymbol{K} = (K_1, \ldots, K_m)$ of a HyperLogLog sketch with parameters p and q are equal to $\boldsymbol{k} = (k_1, \ldots, k_m)$ is given by the corresponding probability mass function

$$\rho(\boldsymbol{k}|n) = \sum_{n_1+\ldots+n_m=n} \binom{n}{n_1, \ldots, n_m} \frac{1}{m^n} \prod_{i=1}^{m} \gamma(k_i|n_i) \tag{2}$$

where n is the cardinality. The n distinct elements are distributed over all m registers according to a multinomial distribution with equal probabilities. $\gamma(k|n)$ is the probability that the value of a register is equal to k, after it was selected n times by the insertion algorithm

$$\gamma(k|n) := \begin{cases} 1 & n = 0 \wedge k = 0 \\ 0 & n = 0 \wedge 1 \le k \le q+1 \\ 0 & n \ge 1 \wedge k = 0 \\ \left(1 - \frac{1}{2^k}\right)^n - \left(1 - \frac{1}{2^{k-1}}\right)^n & n \ge 1 \wedge 1 \le k \le q \\ 1 - \left(1 - \frac{1}{2^q}\right)^n & n \ge 1 \wedge k = q+1 \end{cases}. \tag{3}$$

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_k is the multiplicity of value k. As a consequence, the multiplicity vector $\boldsymbol{C} := (C_0, \ldots, C_{q+1})$ 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 $p = \log_2 \|\boldsymbol{C}\|_1$ and $q = \dim \boldsymbol{C} - 2$, respectively.

2.3. Poisson approximation

Due to the statistical dependence of the register values, the probability mass function (2) is difficult to analyze. Therefore, a Poisson model can be used [6], which assumes that the cardinality itself is distributed according to a Poisson distribution

$$n \sim \text{Poisson}(\lambda). \tag{4}$$

Under the Poisson model the distribution of the register values is

$$\begin{aligned}
\rho(\boldsymbol{k}|\lambda) &= \sum_{n=0}^{\infty} \rho(\boldsymbol{k}|n) e^{-\lambda} \frac{\lambda^n}{n!} && (5)\\
&= \sum_{n_1=0}^{\infty} \cdots \sum_{n_m=0}^{\infty} \prod_{i=1}^{m} \gamma(k_i|n_i) e^{-\frac{\lambda}{m}} \frac{\lambda^{n_i}}{n_i! m^{n_i}} \\
&= \prod_{i=1}^{m} \sum_{n=0}^{\infty} \gamma(k_i|n) e^{-\frac{\lambda}{m}} \frac{\lambda^n}{n! m^n} \\
&= \prod_{k=0}^{q+1} \left(\sum_{n=0}^{\infty} \gamma(k|n) e^{-\frac{\lambda}{m}} \frac{\lambda^n}{n! m^n} \right)^{c_k} \\
&= e^{-c_0 \frac{\lambda}{m}} \left(\prod_{k=1}^{q} \left(e^{-\frac{\lambda}{m 2^k}} \left(1 - e^{-\frac{\lambda}{m 2^k}} \right) \right)^{c_k} \right) \left(1 - e^{-\frac{\lambda}{m 2^q}} \right)^{c_{q+1}} . && (6)
\end{aligned}$$

Here c_k denotes the multiplicity of value k in the multiset $\{k_1, \ldots, k_m\}$. The final factorization shows that under the Poisson values the register values $K_1, \ldots, K_m$ are independent and identically distributed. The probability that a register has a value less than or equal to k for a given rate λ is defined by

$$P(K \le k|\lambda) = \begin{cases} 0 & k < 0 \\ e^{-\frac{\lambda}{m 2^k}} & 0 \le k \le q \\ 1 & k > q \end{cases} . \tag{7}$$

2.4. Depoissonization

Due to the simpler probability mass function, it is easier to find an estimator $\hat{\lambda} = \hat{\lambda}(\boldsymbol{K})$ for the Poisson rate λ rather than for the cardinality n in the fixed-size model (2).

Depoissonization [14] finally allows to translate the estimates back to the fixed-size model. Assume we have found an unbiased estimator for the Poisson rate

$$\mathbb{E}(\hat{\lambda}|\lambda) = \lambda \quad \text{for all } \lambda \geq 0. \tag{8}$$

We know from (5)

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

and therefore

$$\sum_{n=0}^{\infty} \mathbb{E}(\hat{\lambda}|n) e^{-\lambda} \frac{\lambda^n}{n!} = \lambda \tag{10}$$

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

$$\mathbb{E}(\hat{\lambda}|n) = n. \tag{11}$$

Hence, 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 the number of distinct element insertions a register needs to reach the value k is proportional to $m2^k$. Given that, a rough cardinality estimate can be obtained by calculating the average over the values $\{m2^{K_1}, \ldots, m2^{K_m}\}$.

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 [15]. 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 which is given by

$$\hat{n}_{\text{raw}} = \alpha_m \frac{m}{\frac{1}{m2^{K_1}} + \ldots + \frac{1}{m2^{K_m}}} = \frac{\alpha_m m^2}{\sum_{i=1}^{m} 2^{-K_i}} = \frac{\alpha_m m^2}{\sum_{k=0}^{q+1} C_k 2^{-k}}. \tag{12}$$

Here α_m is a bias correction factor which was derived for a given number of registers m to be [6]

$$\alpha_m := \left(m \int_0^{\infty} \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}. \tag{13}$$

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

$$\alpha_\infty := \lim_{m\to\infty} \alpha_m = \frac{1}{2\log 2} \approx 0.7213475, \tag{14}$$

as approximation for α_m in (12), 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 where it fails to return accurate estimates. In order to cover the entire range of cardinalities, corrections for small and large cardinalities have been proposed.

As mentioned in Section 2.1, a HyperLogLog sketch with parameters (p, q) can be mapped to a $(p, 0)$-HyperLogLog sketch. Since $q = 0$ corresponds to linear counting and the reduced HyperLogLog sketch corresponds to a bitset with C_0 zeros, the linear counting cardinality estimator [13] can be used

$$\hat{n}_{\text{small}} = m \log(m/C_0). \tag{15}$$

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 estimator for small cardinalities 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

$$\hat{n}_{\text{large}} = -2^{32} \log\left(1 - \hat{n}_{\text{raw}}/2^{32}\right). \tag{16}$$

The original estimation algorithm as presented in [6] including corrections for small and large cardinalities is summarized by Algorithm 2. The relative estimation error for the original method is shown in Figure 3. Unfortunatley, as can be clearly seen, the ranges where the 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 formula (16) is not satisfying either as it does not reduce the estimation error. Quite the contrary, it even makes the bias worse. However, instead of underestimating the cardinalities, they are now overestimated. Another indication for the incorrectness of the proposed large range correction is the fact that it is not even

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($\boldsymbol{K}$)
  $m \leftarrow \dim \boldsymbol{K}$
  $\hat{n}_{\text{raw}} = \alpha_m m^2 \left(\sum_{i=1}^{m} 2^{-K_i}\right)^{-1}$ ▷ raw estimate (12)
  if $\hat{n}_{\text{raw}} \leq \frac{5}{2}m$ **then**
    $C_0 = |\{i|K_i = 0\}|$
    if $C_0 \neq 0$ **then**
      return $m \log(m/C_0)$ ▷ small range correction (15)
    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 (16)
  end if
end function

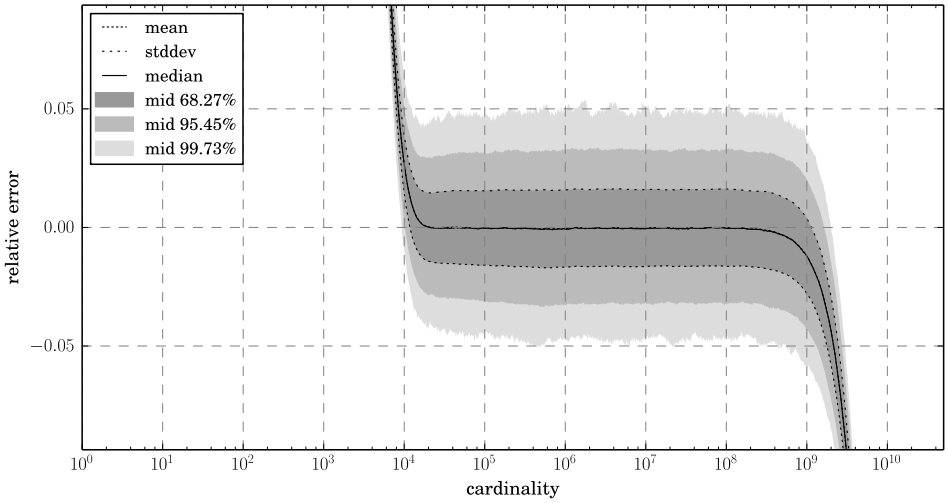

Figure 1: The distribution of the relative estimation error over the cardinality for the raw estimator after evaluation of 10,000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$.

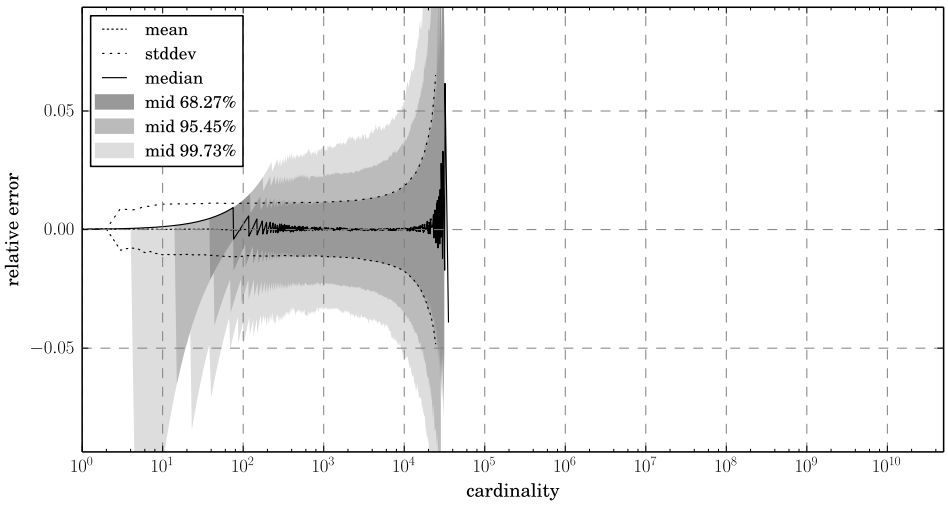

Figure 2: The distribution of the relative estimation error for the linear counting estimator after evaluation of 10,000 randomly generated bitmaps of size 2^{12} which correspond to HyperLogLog sketches with parameters $p = 12$ and $q = 0$.

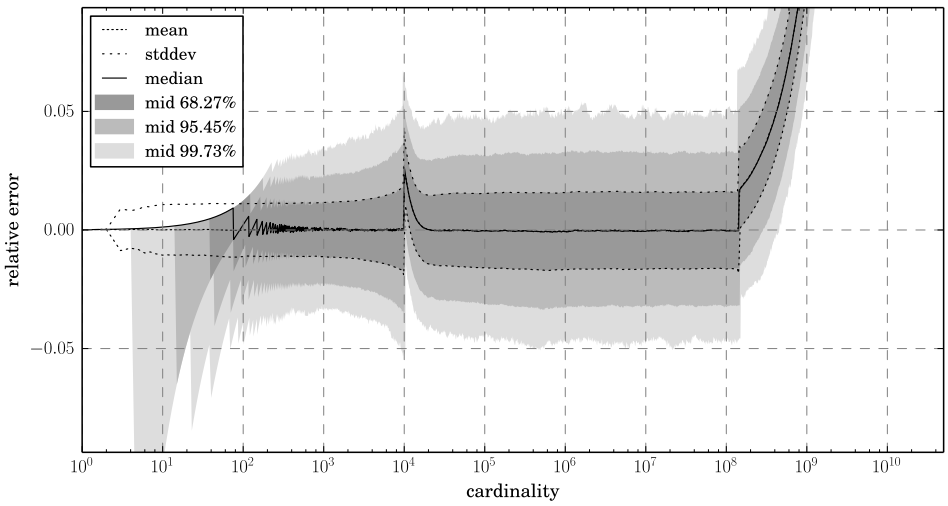

Figure 3: The distribution of the relative estimation error over the cardinality for the original estimation algorithm after evaluation of 10,000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$.

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 formula.

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 easily accomplished by using hash values with more bits ($p + q > 32$). If $q \geq 30$, 5-bit registers are no longer sufficient to represent all possible values in the range $[0, q+1]$. If, for example, 64-bit hash values ($p+q = 64$) are used, as proposed in [7], 6 bits per register are needed. Hash value sizes with more than 64 bits are useless in practice, because it is unrealistic to encounter cardinalities of order 2^{64} for which the raw estimator would be biased.

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 and simple derivation without the restriction tolarge cardinalities ($n \to \infty$) and without using complex analysis as in [6].

Let us assume that the register values have following cumulative distribution function

$$P(K \leq k|\lambda) = e^{-\frac{\lambda}{m2^k}}. \tag{17}$$

For now we ignore that this distribution has infinite support and differs from the register value distribution under the Poisson model (7), whose support is limited to the range $[0, q+1]$. For a random variable K obeying (17) the expectation of 2^{-K} is given by

$$\mathbb{E}(2^{-K}) = \sum_{k=-\infty}^{\infty} 2^{-k} \left(e^{-\frac{\lambda}{m2^k}} - e^{-\frac{\lambda}{m2^{k-1}}} \right) = \frac{1}{2} \sum_{k=-\infty}^{\infty} 2^k e^{-\frac{\lambda}{m}2^k} = \frac{\alpha_\infty \, m \, \xi(\log_2(\lambda/m))}{\lambda}, \tag{18}$$

where the function

$$\xi(x) := \log(2) \sum_{k=-\infty}^{\infty} 2^{k+x} e^{-2^{k+x}} \tag{19}$$

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 (17). For large sample sizes $m \to \infty$ we have asymptotically

$$\mathbb{E}\left(\frac{1}{2^{-K_1} + \ldots + 2^{-K_m}} \right) \underset{m\to\infty}{=} \frac{1}{\mathbb{E}(2^{-K_1} + \ldots + 2^{-K_m})} = \frac{1}{m\mathbb{E}(2^{-K})}. \tag{20}$$

Together with (18) we obtain

$$\lambda = \mathbb{E}\left(\frac{\alpha_\infty \, m^2 \, \xi(\log_2(\lambda/m))}{2^{-K_1} + \ldots + 2^{-K_m}} \right) \quad \text{for } m \to \infty. \tag{21}$$

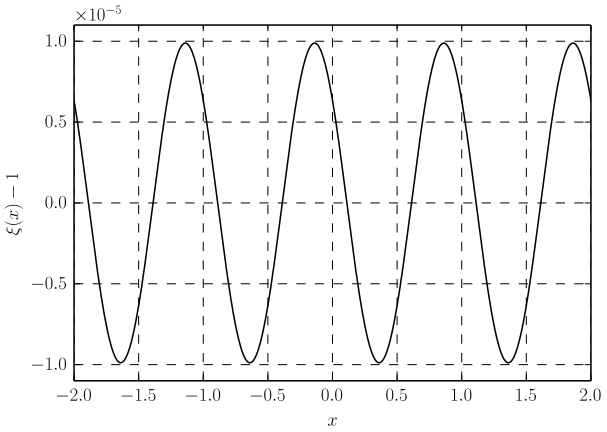

Figure 4: The deviation of $\xi(x)$ from 1.

Therefore, the asymptotic relative bias of

$$\hat{\lambda} = \frac{\alpha_\infty \, m^2}{2^{-K_1} + \ldots + 2^{-K_m}} \tag{22}$$

is bounded by ε_ξ, which makes this statistic a good estimator for the Poisson parameter. It also corresponds to the raw estimator (12), if the Poisson parameter estimate is used as cardinality estimate (see Section 2.4).

3.2. Limitations of the raw estimator

The raw estimator is based on two prerequisites. First, the number of registers needs to be sufficiently large ($m \to \infty$). And second, the distribution of register values can be described by (17). However, the latter is not true for small and large cardinalities, which is finally the reason for the bias of the raw estimator.

A random variable K' with cumulative distribution function (17) can be transformed into a random variable K with cumulative distribution function (7) using

$$K = \min\big(\max\big(K', 0\big)\,, q + 1\big)\,. \tag{23}$$

Therefore, the register values $K_1, \ldots, K_m$ can be seen as the result after applying this transformation to a sample $K'_1, \ldots, K'_m$ of the distribution described by (17). If all registers values of the HyperLogLog sketch lie in the range $[1, q]$, they are identical to the values $K'_1, \ldots, K'_m$. In other words, the observed register values are also a plausible sample of the assumed distribution described by (17) in this case. Hence, as long as all or at least most register values are in the range $[1, q]$ the approximation of (7) by (17) is valid. This explains why the raw estimator works best for intermediate cardinalities. However, for small and large cardinalities many registers have values equal to 0 or $q+1$, respectively, which cannot be described by (17) and which finally leads to the observed bias.

3.3. Corrections to the raw estimator

If we knew the values $K'_1, \ldots, K'_m$ for which the transformation (23) led to the register values $K_1, \ldots, K_m$, we would be able to estimate λ using

$$\hat{\lambda} = \frac{\alpha_\infty \, m^2}{2^{-K'_1} + \ldots + 2^{-K'_m}}. \tag{24}$$

As we have already shown, this estimator is approximately unbiased, because all K'_i follow the assumed distribution. It would be even sufficient, if we know the multiplicity C'_k of each $k \in \mathbb{Z}$ in $\{K'_1, \ldots, K'_m\}$, $C'_k := |\{i | k = K'_i\}|$, because the raw estimator can be also written as

$$\hat{\lambda} = \frac{\alpha_\infty \, m^2}{\sum_{k=-\infty}^{\infty} C'_k 2^{-k}}. \tag{25}$$

Due to (23), the multiplicities C'_k and the multiplicities C_k for the register values have following relationships

$$\begin{aligned} C_0 &= \textstyle\sum_{k=-\infty}^{0} C'_k, \\ C_k &= C'_k, \quad 1 \leq k \leq q, \\ C_{q+1} &= \textstyle\sum_{k=q+1}^{\infty} C'_k. \end{aligned} \tag{26}$$

The idea is now to find estimates $\hat{c}'_k$ for all $k \in \mathbb{Z}$ and use them as replacements for C'_k in (25). For $k \in [1, q]$ where $C_k = C'_k$ we can use the trivial estimators

$$\hat{c}'_k := C_k, \quad 1 \leq k \leq q. \tag{27}$$

To get estimators for $k \leq 0$ and $k \geq q + 1$, we consider the expectations

$$\mathbb{E}(C'_k) = m e^{-\frac{\lambda}{m 2^k}} \left(1 - e^{-\frac{\lambda}{m 2^k}}\right). \tag{28}$$

From (7) we know that $\mathbb{E}(C_0/m) = e^{-\frac{\lambda}{m}}$ and $\mathbb{E}(1 - C_{q+1}/m) = e^{-\frac{\lambda}{m 2^q}}$, and therefore, we can also write

$$\mathbb{E}(C'_k) = m \left(\mathbb{E}(C_0/m)\right)^{2^{-k}} \left(1 - \left(\mathbb{E}(C_0/m)\right)^{2^{-k}}\right) \tag{29}$$

and

$$\mathbb{E}(C'_k) = m \left(\mathbb{E}(1 - C_{q+1}/m)\right)^{2^{q-k}} \left(1 - \left(\mathbb{E}(1 - C_{q+1}/m)\right)^{2^{q-k}}\right), \tag{30}$$

which motivates us to use

$$\hat{c}'_k = m \left(C_0/m\right)^{2^{-k}} \left(1 - \left(C_0/m\right)^{2^{-k}}\right) \tag{31}$$

as estimator for $k \leq 0$ and

$$\hat{c}'_k = m \left(1 - C_{q+1}/m\right)^{2^{q-k}} \left(1 - \left(1 - C_{q+1}/m\right)^{2^{q-k}}\right) \tag{32}$$

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

Inserting all these estimators into (25) as replacements for C'_k finally gives

$$\hat{\lambda} = \frac{\alpha_\infty m^2}{\sum_{k=-\infty}^{\infty} \hat{c}'_k 2^{-k}} = \frac{\alpha_\infty m^2}{m\,\sigma(C_0/m) + \sum_{k=1}^{q} C_k 2^{-k} + m\,\tau(1 - C_{q+1}/m)2^{-q}} \tag{33}$$

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 (12), respectively. The functions σ and τ are defined as

$$\sigma(x) := x + \sum_{k=1}^{\infty} x^{2^k} 2^{k-1} \tag{34}$$

and

$$\tau(x) := \sum_{k=1}^{\infty} x^{2^{-k}} \left(1 - x^{2^{-k}}\right) 2^{-k}. \tag{35}$$

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 $q = 0$

$$\hat{\lambda} = \frac{\alpha_\infty m}{\sigma(C_0/m) + \tau(C_0/m)} = \frac{m \log(m/C_0)}{\xi(\log_2(\log(m/C_0)))} \approx m \log(m/C_0)\,, \tag{36}$$

which is almost identical to the linear counting estimator used for the small range correction (15). Here we used again the fact that the function ξ can be well approximated by 1 (see Section 3.1).

3.4. Corrected raw estimation algorithm

The corrected raw estimator (33) 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 of 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 $m + 1$.

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 uniform hash function, element hash values can be mocked by random numbers. For the following results we used the Mersenne Twister random number generator with 19,937 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).

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

function EstimateCardinality($\boldsymbol{C}$)
  $m \leftarrow \|\boldsymbol{C}\|_1$
  $z \leftarrow m \cdot \tau(1 - C_{q+1}/m)$ ▷ 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 $\alpha_\infty m^2/z$
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

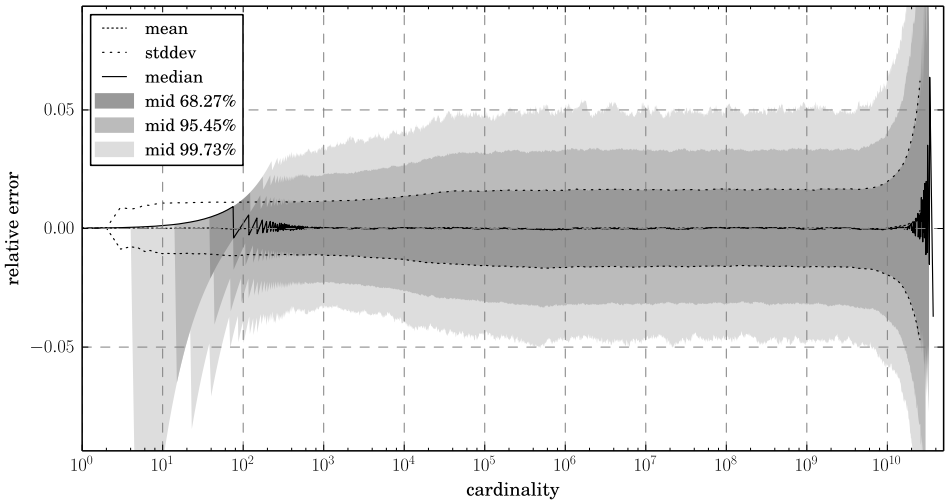

Figure 5: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

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 HyperLogLog configuration with parameters $p = 12$ and $q = 52$. 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.

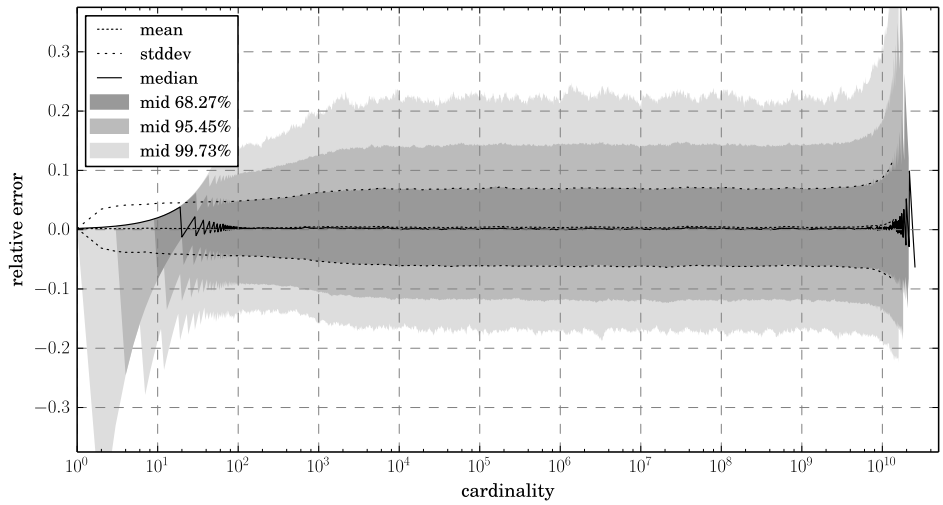

Figure 6: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

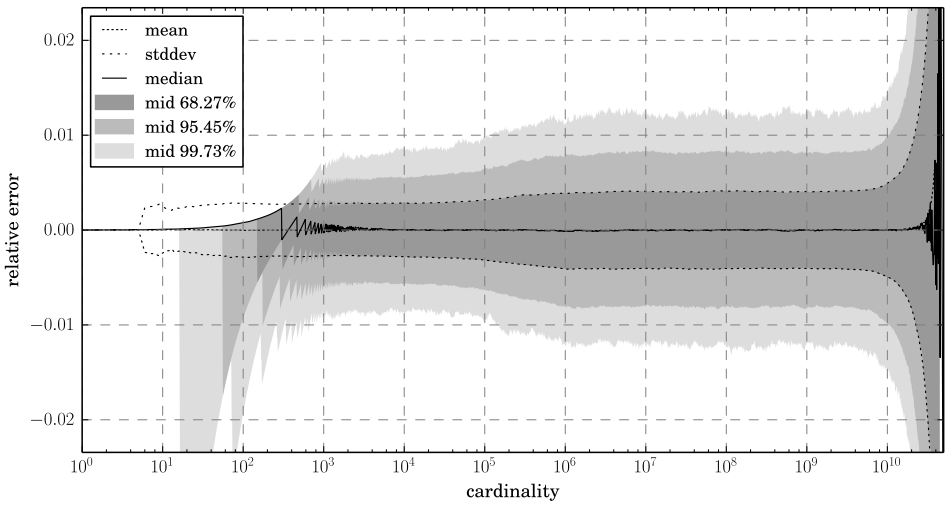

Figure 7: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

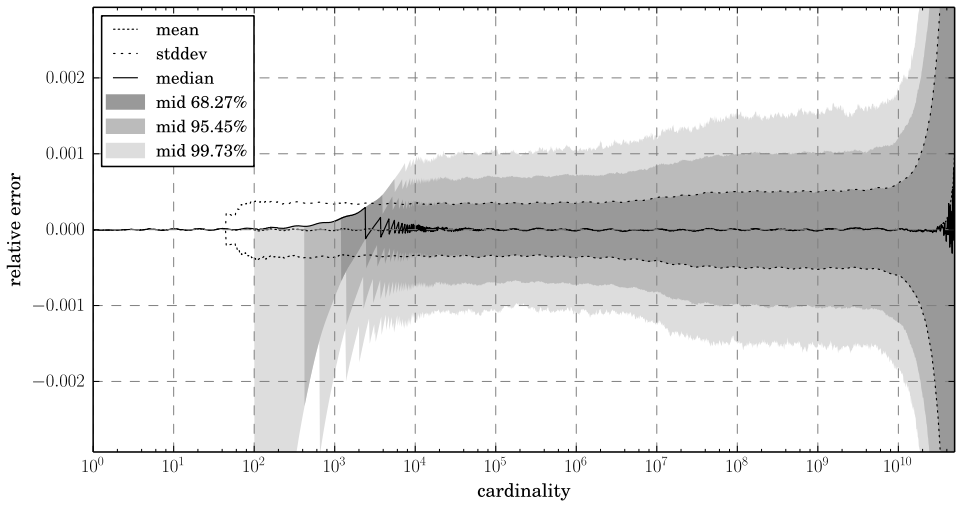

Figure 8: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 10$.

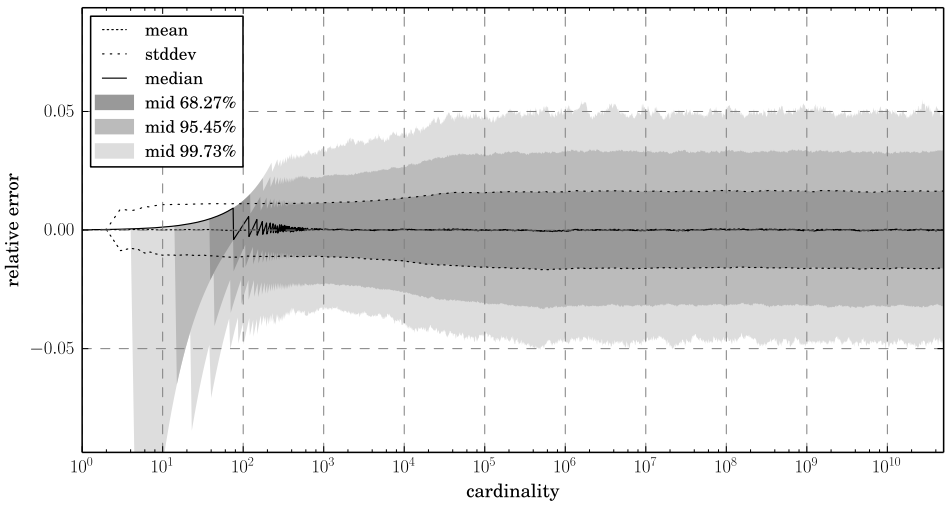

Figure 9: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 52$.

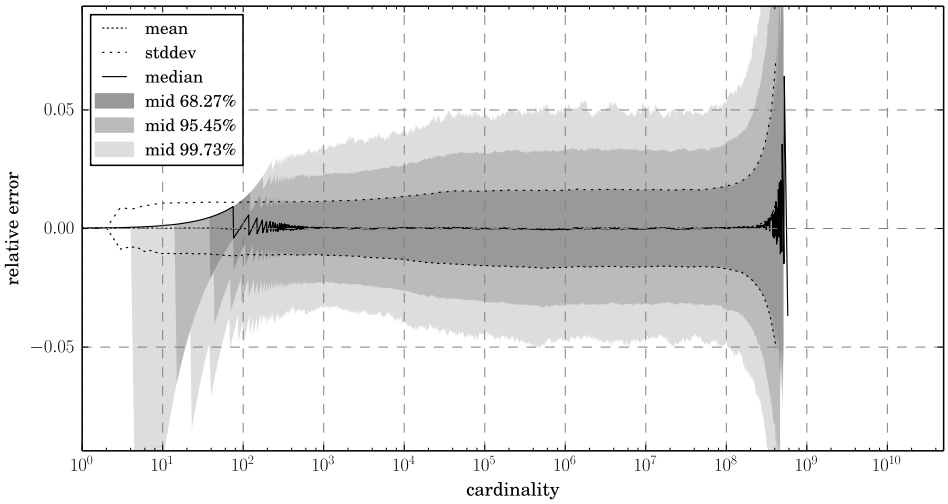

Figure 10: Relative error of the corrected raw estimator as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 14$.

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 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

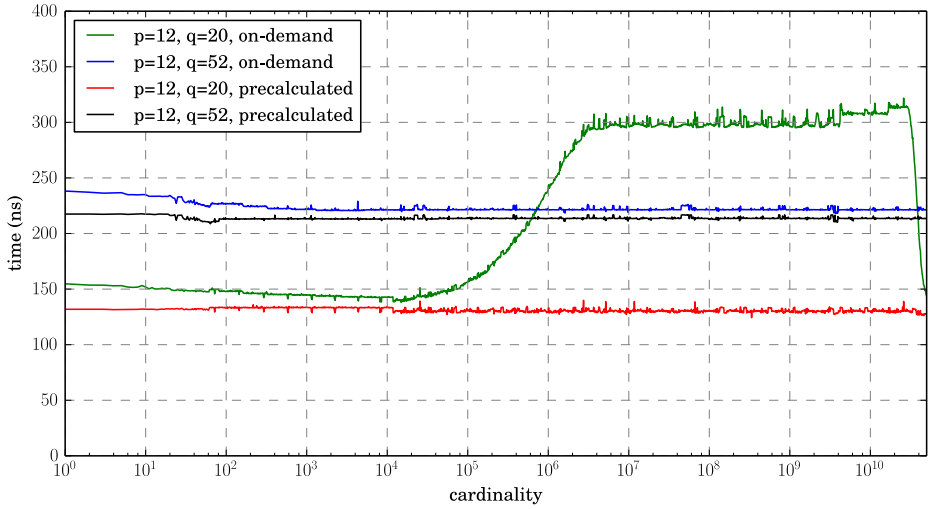

Figure 11: Average computation time as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3 GHz when estimating the cardinality from HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively. Both cases, σ and τ precalculated and calculated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

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. If we consider a HyperLogLog sketch with precision $p = 12$ which uses 5 bits per register, the total data size of the HyperLogLog sketch is 2.5 kB minimum. Consequently, the 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 [16]. 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 remark that we are not the first applying the maximum likelihood method to HyperLogLog sketches [17], where the case without stochastic averaging (compare Section 2.1) was considered. The register values are statistically independent by nature in this case which allows factorization of the joint probability mass function and allows a straightforward application of the maximum likelihood method.

The more relevant case with stochastic averaging, which is considered in this paper, is more complicated. However, with the help of the Poisson approximation, we are able to still apply the maximum likelihood method and 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 estimator (12) was first found after several attempts using the geometric mean [6, 15].

4.1. Log-likelihood function

Using the probability mass function of the Poisson model (6) the log-likelihood and its derivative are given by

$$\log \mathcal{L}(\lambda|\boldsymbol{K}) = -\frac{\lambda}{m}\sum_{k=0}^{q}\frac{C_k}{2^k} + \sum_{k=1}^{q} C_k \log\Big(1 - e^{-\frac{\lambda}{m2^k}}\Big) + C_{q+1}\log\Big(1 - e^{-\frac{\lambda}{m2^q}}\Big) \tag{37}$$

and

$$\frac{d}{d\lambda}\log \mathcal{L}(\lambda|\boldsymbol{K}) = -\frac{1}{\lambda}\left(\frac{\lambda}{m}\sum_{k=0}^{q}\frac{C_k}{2^k} + \sum_{k=1}^{q} C_k \frac{\frac{\lambda}{m2^k}}{1 - e^{\frac{\lambda}{m2^k}}} + C_{q+1}\frac{\frac{\lambda}{m2^q}}{1 - e^{\frac{\lambda}{m2^q}}}\right). \tag{38}$$

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

$$\hat{\lambda} = m\hat{x}, \tag{39}$$

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

$$f(x) := x\sum_{k=0}^{q}\frac{C_k}{2^k} + \sum_{k=1}^{q} C_k \frac{\frac{x}{2^k}}{1 - e^{\frac{x}{2^k}}} + C_{q+1}\frac{\frac{x}{2^q}}{1 - e^{\frac{x}{2^q}}}. \tag{40}$$

This function can also be written as

$$f(x) := x\sum_{k=0}^{q}\frac{C_k}{2^k} + \sum_{k=1}^{q} C_k h\Big(\frac{x}{2^k}\Big) + C_{q+1}h\Big(\frac{x}{2^q}\Big) - (m - C_0)\,, \tag{41}$$

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

$$h(x) := 1 - \frac{x}{e^x - 1}. \tag{42}$$

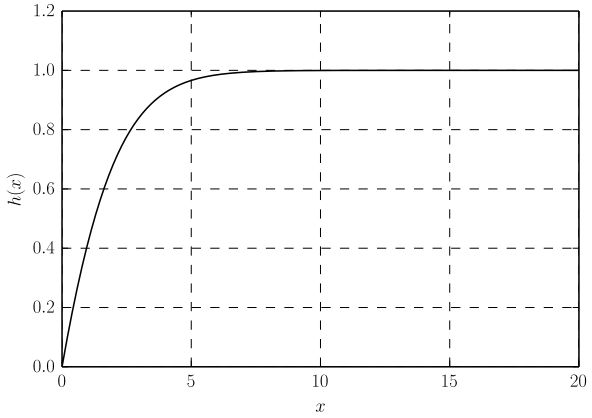

Figure 12: The function $h(x)$.

$h(x)$ 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 infinite.

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 (41) as

$$f(x) := C_0 x + C_1 \left(h\Big(\frac{x}{2^1}\Big) + \frac{x}{2^1} - 1 \right) + C_2 \left(h\Big(\frac{x}{2^2}\Big) + \frac{x}{2^2} - 1 \right) + \ldots \\ \cdots + C_q \left(h\Big(\frac{x}{2^q}\Big) + \frac{x}{2^q} - 1 \right) + C_{q+1} \left(h\Big(\frac{x}{2^q}\Big) - 1 \right). \quad (43)$$

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 prior the update.

For the special case $q = 0$, which corresponds to the already mentioned linear counting algorithm, (40) can be solved analytically. In this case, the maximum likelihood method under the Poisson model leads directly to the linear counting estimator (15). 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 (41) gives an upper bound for $f(x)$:

$$f(x) \leq x \sum_{k=0}^{q} \frac{C_k}{2^k} + (m - C_0) \cdot h\left(x \cdot \frac{\sum_{k=1}^{q} \frac{C_k}{2^k} + \frac{C_{q+1}}{2^q}}{m - C_0} \right) - (m - C_0)\,. \tag{44}$$

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

$$\hat{x} \geq \frac{m - C_0}{\sum_{k=1}^{q} \frac{C_k}{2^k} + \frac{C_{q+1}}{2^q}} \log\left(1 + \frac{\sum_{k=1}^{q} \frac{C_k}{2^k} + \frac{C_{q+1}}{2^q}}{\sum_{k=0}^{q} \frac{C_k}{2^k}} \right). \tag{45}$$

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

$$\hat{x} \geq \frac{m - C_0}{C_0 + \frac{3}{2} \sum_{k=1}^{q} \frac{C_k}{2^k} + \frac{C_{q+1}}{2^q}}. \tag{46}$$

Using the monotonicity of h, the lower bound

$$f(x) \geq x \sum_{k=0}^{q} \frac{C_k}{2^k} + \sum_{k=1}^{q} C_k h\Big(\frac{x}{2^{K'_{\max}}} \Big) + C_{q+1} h\Big(\frac{x}{2^{K'_{\max}}} \Big) - (m - C_0) \tag{47}$$

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

$$\hat{x} \leq 2^{K'_{\max}} \log\left(1 + \frac{m - C_0}{2^{K'_{\max}} \sum_{k=0}^{q} \frac{C_k}{2^k}} \right) \tag{48}$$

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

$$\hat{x} \leq \frac{m - C_0}{\sum_{k=0}^{q} \frac{C_k}{2^k}}. \tag{49}$$

If the HyperLogLog sketch is in the intermediate range, where $C_0 = C_{q+1} = 0$ the bounds (46) and (49) differ only by a constant factor from the raw estimator (12). Hence, the maximum likelihood method leads directly to the harmonic mean 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, will 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

$$x_i = x_{i-1} - (x_{i-1} - x_{i-2}) \frac{f(x_{i-1})}{f(x_{i-1}) - f(x_{i-2})} \tag{50}$$

If $x_0 = 0$ with $f(x_0) = -(m - C_0)$, and x_1 is equal to one of the derived lower bounds (45) or (46), the sequence $(x_0, x_1, x_2, \ldots)$ is montone increasing. Using the definitions

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

and

$$g(x) := f(x) + (m - C_0) = x \sum_{k=0}^{q} \frac{C_k}{2^k} + \sum_{k=1}^{q} C_k h\Big(\frac{x}{2^k}\Big) + C_{q+1} h\Big(\frac{x}{2^q}\Big) \tag{52}$$

the iteration scheme can also be written as

$$\Delta x_i = \Delta x_{i-1} \frac{(m - C_0) - g(x_{i-1})}{g(x_{i-1}) - g(x_{i-2})} \tag{53}$$

$$x_i = x_{i-1} + \Delta x_i \tag{54}$$

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 be presented 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 (52). 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 sum $\sum_{k=0}^{q} \frac{C_k}{2^k}$ in (52) 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 $\left\{ \frac{x}{2^{K'_{\max}}}, \frac{x}{2^{K'_{\max}-1}}, \ldots, \frac{x}{2^{K_{\min}}} \right\}$ where $K'_{\max} := \min(K_{\max}, q)$. This series corresponds to a geometric series with ratio 2. A straightforward calculation using (42) is very expensive because of the

exponential function. However, if we know $h\left(\frac{x}{2^{K'_{\max}}}\right)$ all other required function values can be easily obtained using the identity

$$h(4x) = \frac{x + h(2x)\,(1 - h(2x))}{x + (1 - h(2x))}. \tag{55}$$

This recursive formula is stable in a sense that the relative error of $h(4x)$ is smaller than that of $h(2x)$ as shown in Appendix A.

- If x is smaller than 0.5, the function $h(x)$ can be well approximated by a Taylor series around $x = 0$

$$h(x) = \frac{x}{2} - \frac{x^2}{12} + \frac{x^4}{720} - \frac{x^6}{30240} + \mathcal{O}(x^8) \tag{56}$$

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

$$h(x) = x' - x''/3 + \left(x''x''\right)\left(1/45 - x''/472.5\right) + \mathcal{O}(x^8) \tag{57}$$

The smallest argument for which h needs to be evaluated is $\frac{x}{2^{K'_{\max}}}$. If $C_{q+1} = 0$, we can find an upper bound for the smallest argument using (48)

$$\frac{x}{2^{K'_{\max}}} \leq \log\left(1 + \frac{\sum_{k=0}^{K'_{\max}} C_k}{2^{K'_{\max}} \sum_{k=0}^{K'_{\max}} \frac{C_k}{2^k}}\right) \leq \log 2 \approx 0.693. \tag{58}$$

In practice, $\frac{x}{2^{K'_{\max}}} \leq 0.5$ is satisfied most of the time as long as only a few registers are saturated ($C_{q+1} \ll m$). In case $\frac{x}{2^{K'_{\max}}} > 0.5$, $h\left(\frac{x}{2^\kappa}\right)$ is calculated instead with $\kappa = 2 + \lfloor \log_2(x) \rfloor$. By definition, $\frac{x}{2^\kappa} \leq 0.5$ which allows using the Taylor series approximation. $h\left(\frac{x}{2^{K'_{\max}}}\right)$ is finally obtained after $\kappa - K'_{\max}$ iterations using (55). As shown in Appendix B, a small approximation error of h does not have much impact on the error of the maximum likelihood estimate as long as most registers are not yet saturated.

Putting all these optimizations together finally gives the new cardinality estimation algorithm as presented in Algorithm 4. The algorithm requires only elementary operations, except for large cardinalities, where it is worth to take the strong (45) instead of the weak lower bound (46) as initial value, which requires a single logarithm evaluation. The stronger bound is a much better approximation especially for large cardinalities and substantially reduces the number of required iteration cycles.

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

Algorithm 4 Maximum likelihood cardinality estimation.

function EstimateCardinality($\boldsymbol{C}$)
  $q \leftarrow \dim(\boldsymbol{C}) - 2$
  $K_{\min} \leftarrow \min\{k | C_k > 0\}$
  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
  ▷ here $z = \sum_{k=1}^{q} \frac{C_k}{2^k}$
  $m \leftarrow m' + C_0$
  $c \leftarrow C_{q+1}$
  if $q \geq 1$ **then**
    $c \leftarrow c + C_{K'_{\max}}$
  end if
  $g_{\text{prev}} \leftarrow 0$
  $a \leftarrow z + C_0$ ▷ $a = \sum_{k=0}^{q} \frac{C_k}{2^k}$
  $b \leftarrow z + C_{q+1} \cdot 2^{-q}$ ▷ $b = \sum_{k=1}^{q} \frac{C_k}{2^k} + \frac{C_{q+1}}{2^q}$
  if $b \leq 1.5 \cdot a$ **then**
    $x \leftarrow m'/(0.5 \cdot b + a)$ ▷ weak lower bound (46)
  else
    $x \leftarrow m'/b \cdot \log(1 + b/a)$ ▷ strong lower bound (45)
  end if

Algorithm 4 Maximum likelihood cardinality estimation (continued).

$\Delta x \leftarrow x$
while $\Delta x > x \cdot \varepsilon$ **do** ▷ secant method iteration, $\varepsilon = 10^{-2}$ (see Section 4.3)
$\kappa \leftarrow 2 + \lfloor \log_2(x) \rfloor$
$x' \leftarrow x \cdot 2^{-\max(K'_{\max},\kappa)-1}$ ▷ $x' \in [0, 0.25]$
$x'' \leftarrow x' \cdot x'$
$h \leftarrow x' - x''/3 + (x'' \cdot x'') \cdot (1/45 - x''/472.5)$ ▷ Taylor approximation (57)
for $k \leftarrow (\kappa - 1), K'_{\max}$ **do**
$h \leftarrow \frac{x'+h\cdot(1-h)}{x'+(1-h)}$ ▷ calculate $h\left(\frac{x}{2^k}\right)$, see (55), at this point $x' = \frac{x}{2^{k+2}}$
$x' \leftarrow 2x'$
end for
$g \leftarrow c \cdot h$ ▷ compare (52)
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 (55), 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 (53)
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

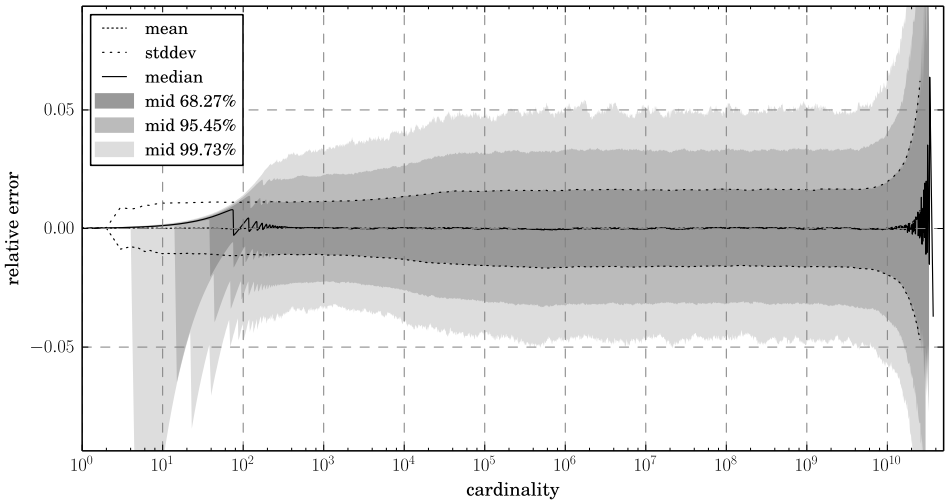

Figure 13: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

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

While the union of two sets that are represented by HyperLogLog sketches can be straightforwardly computed (1), the computation of cardinalities of other set operations like intersections and differences is more challenging. A common approach to calculate

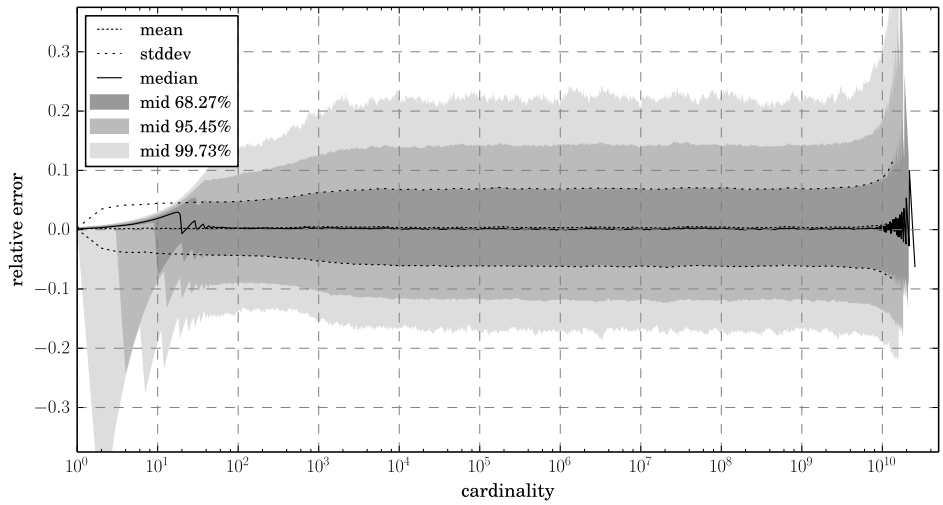

Figure 14: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

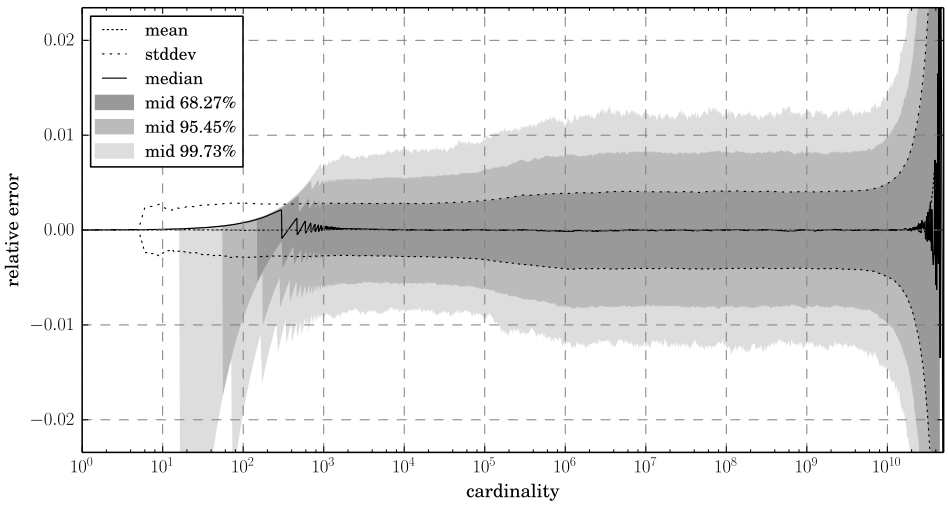

Figure 15: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

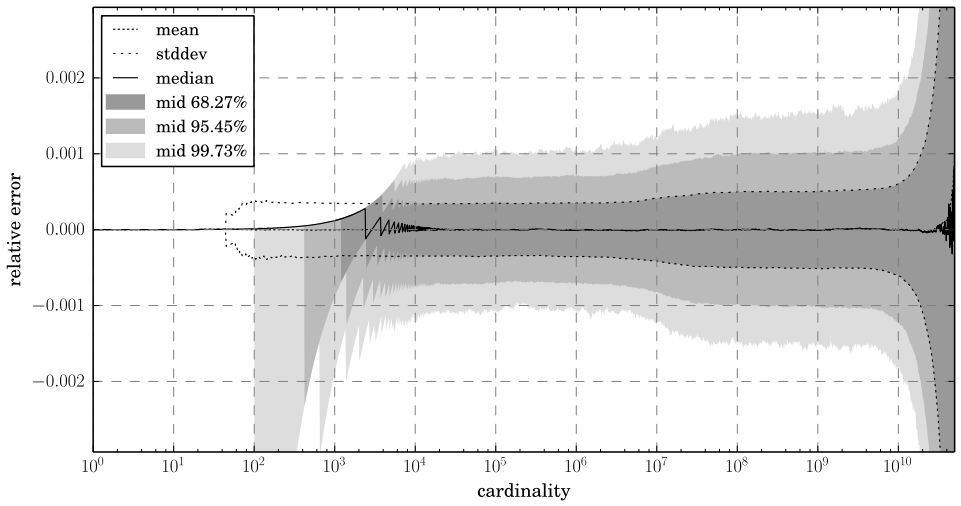

Figure 16: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 10$.

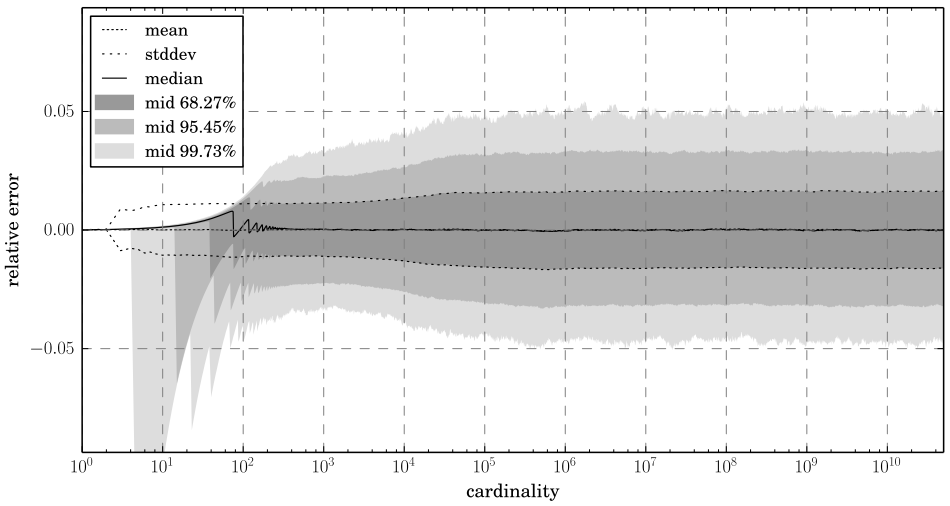

Figure 17: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 52$.

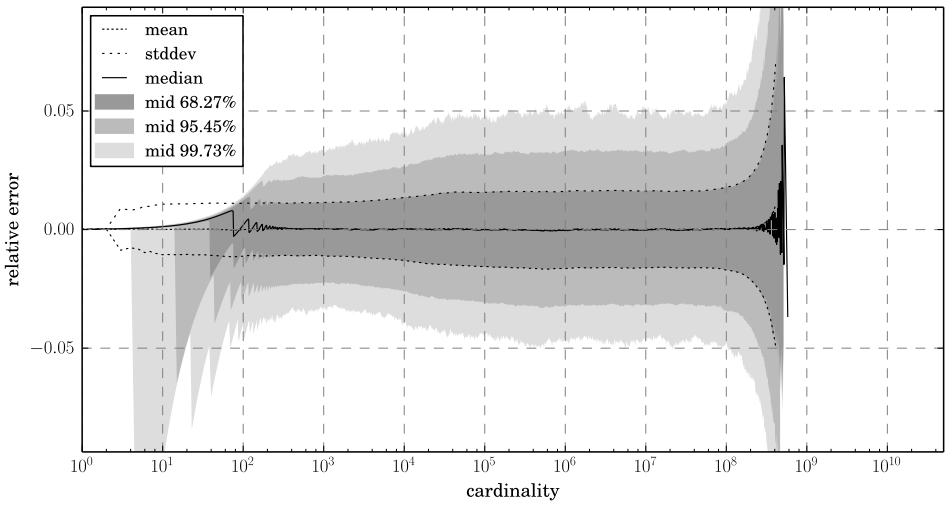

Figure 18: Relative error of the maximum likelihood estimates as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 14$.

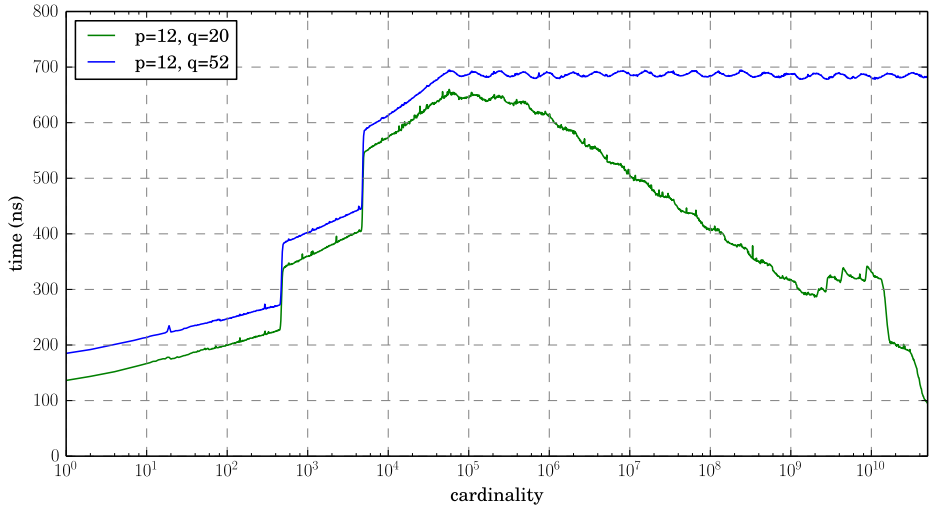

Figure 19: Average execution time of the maximum likelihood estimation algorithm as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3 GHz for HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively.

the cardinality of intersections is the inclusion-exclusion principle

$$|S_1 \cap S_2| = |S_1| + |S_2| - |S_1 \cup S_2| \,. \tag{59}$$

Unfortunately, this approach can be very inaccurate. Due to estimation errors of HyperLogLog sketches the result of (59) can be even negative.

Motivated by the good results we have obtained for a single HyperLogLog sketch using the maximum likelihood method in combination with the Poisson approximation, we are tempted to also use it for the estimation of set operations. Assume two given HyperLogLog sketches with register values $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ representing the sets S_1 and S_2, respectively. The goal is to find estimates for the cardinalities of the pair-wise disjoint sets $X = S_1 \cap S_2$, $A = S_1 \setminus S_2$, and $B = S_2 \setminus S_1$.

The Poisson approximation allows us to assume that the elements of A are inserted into the HyperLogLog sketch representing S_1 at rate λ_A. Similarly, the elements of B are inserted into the HyperLogLog sketch representing S_2 at rate λ_B. Furthermore, we assume that elements of X are inserted into both HyperLogLog sketches simultaneously at rate λ_X. The goal is now to find estimates, $\hat{\lambda}_A$, $\hat{\lambda}_B$, and $\hat{\lambda}_X$ for the rates which could also be good estimates for the wanted cardinalities of A, B, and X.

5.1. Joint log-likelihood funtion

In order to get maximum-likelihood estimators for $\hat{\lambda}_A$, $\hat{\lambda}_B$, and $\hat{\lambda}_X$ we need to derive the joint probability distribution of two HyperLogLog sketches. 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 can be thought to be constructed from two HyperLogLog sketches representing A and X and merging both using (1). Analogously, the HyperLogLog sketch for S_2 could have been obtained from sketches for B and X. Let K_a, K_b, and K_x be the value of the considered register in the HyperLogLog sketch representing A, B, and X, respectively. The corresponding values in sketches for S_1 and S_2 are given by

$$K_1 = \max(K_a, K_x) \,, \quad K_2 = \max(K_b, K_x) \,. \tag{60}$$

Their joint cumulative probability function is given as

$$\begin{aligned} P(K_1 \le k_1 \wedge K_2 \le k_2) &= P(\max(K_a, K_x) \le k_1 \wedge \max(K_b, K_x) \le k_2) \\ &= P(K_a \le k_1 \wedge K_b \le k_2 \wedge K_x \le \min(k_1, k_2)) \\ &= P(K_a \le k_1)\, P(K_b \le k_2)\, P(K_x \le \min(k_1, k_2)) \,. \end{aligned} \tag{61}$$

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 (7). If we take into account 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

$$P(K_1 \leq k_1 \wedge K_2 \leq k_2) = \begin{cases} 0 & k_1 < 0 \vee k_2 < 0 \\ e^{-\frac{\lambda_a}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}} - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}} & 0 \leq k_1 \leq q \wedge 0 \leq k_2 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}} & 0 \leq k_2 \leq q < k_1 \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}} & 0 \leq k_1 \leq q < k_2 \\ 1 & q < k_1 \wedge q < k_2 \end{cases} \tag{62}$$

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

$$\begin{aligned} \rho(k_1, k_2) = P(K_1 \leq k_1 \wedge K_2 \leq k_2) - P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2) \\ - P(K_1 \leq k_1 \wedge K_2 \leq k_2 - 1) + P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2 - 1), \end{aligned} \tag{63}$$

which finally gives

$$\rho(k_1, k_2) = \begin{cases} e^{-\frac{\lambda_a+\lambda_x}{m} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 0 = k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_b}{m2^q}}\right) & 0 = k_1 < k_2 = q+1 \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right) \left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 1 \leq k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right) \left(1 - e^{-\frac{\lambda_b}{m2^q}}\right) & 1 \leq k_1 < k_2 = q+1 \\ e^{-\frac{\lambda_b+\lambda_x}{m} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 0 = k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_a}{m2^q}}\right) & 0 = k_2 < k_1 = q+1 \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right) \left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 1 \leq k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right) \left(1 - e^{-\frac{\lambda_a}{m2^q}}\right) & 1 \leq k_2 < k_1 = q+1 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m}} & 0 = k_1 = k_2 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^k}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^k}} - e^{-\frac{\lambda_b+\lambda_x}{m2^k}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^k}}\right) & 1 \leq k_1 = k_2 = k \leq q \\ 1 - e^{-\frac{\lambda_a+\lambda_x}{m2^q}} - e^{-\frac{\lambda_b+\lambda_x}{m2^q}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^q}} & k_1 = k_2 = q+1 \end{cases} \tag{64}$$

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

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

$$\begin{aligned}\log(\rho(k_1,k_2)) = &-\frac{\lambda_a}{m2^{k_1}}\left[k_1 \le q\right] - \frac{\lambda_b}{m2^{k_2}}\left[k_2 \le q\right] - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}\left[k_1 \le q \vee k_2 \le q\right] \\ &+ \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right)\left[1 \le k_1 < k_2\right] \\ &+ \log\left(1 - e^{-\frac{\lambda_a}{m2^{\min(k_1,q)}}}\right)\left[k_2 < k_1\right] \\ &+ \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right)\left[1 \le k_2 < k_1\right] \\ &+ \log\left(1 - e^{-\frac{\lambda_b}{m2^{\min(k_2,q)}}}\right)\left[k_1 < k_2\right] \\ &+ \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k_1,q)}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}}\right)\left[1 \le k_1 = k_2\right].\end{aligned} \tag{65}$$

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

$$\rho(\boldsymbol{k}_1, \boldsymbol{k}_2) = \prod_{i=1}^{m}\prod_{j=1}^{m}\rho(k_{1i}, k_{2j}). \tag{66}$$

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

$$\log\mathcal{L}(\lambda_a, \lambda_b, \lambda_x|\boldsymbol{K}_1, \boldsymbol{K}_2) = \sum_{i=1}^{m}\sum_{j=1}^{m}\log(\rho(K_{1i}, K_{2j})) \tag{67}$$

Insertion of (65) results in the generalization of (37) for two HyperLogLog sketches

$$\begin{aligned}\log\mathcal{L}(\lambda_a, \lambda_b, \lambda_x|\boldsymbol{K}_1, \boldsymbol{K}_2) = &-\frac{\lambda_a}{m}\sum_{k=0}^{q}\frac{C_{1k}^{<} + C_k^{=} + C_{1k}^{>}}{2^k} - \frac{\lambda_b}{m}\sum_{k=0}^{q}\frac{C_{2k}^{<} + C_k^{=} + C_{2k}^{>}}{2^k} \\ &- \frac{\lambda_x}{m}\sum_{k=0}^{q}\frac{C_{1k}^{<} + C_k^{=} + C_{2k}^{<}}{2^k} \\ &+ \sum_{k=1}^{q}\log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^k}}\right)C_{1k}^{<} + \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^k}}\right)C_{2k}^{<} \\ &+ \sum_{k=1}^{q+1}\log\left(1 - e^{-\frac{\lambda_a}{m2^{\min(k,q)}}}\right)C_{1k}^{>} + \log\left(1 - e^{-\frac{\lambda_b}{m2^{\min(k,q)}}}\right)C_{2k}^{>} \\ &+ \sum_{k=1}^{q+1}\log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k,q)}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k,q)}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{\min(k,q)}}}\right)C_k^{=},\end{aligned} \tag{68}$$

where the constants $C_{1k}^{>}$, $C_{1k}^{<}$, $C_{2k}^{>}$, $C_{2k}^{<}$, and $C_{k}^{=}$ are defined as

$$\begin{aligned} C_{1k}^{>} &:= |\{i|k = K_{1i} > K_{2i}\}|\,, \\ C_{1k}^{<} &:= |\{i|k = K_{1i} < K_{2i}\}|\,, \\ C_{2k}^{>} &:= |\{i|k = K_{2i} > K_{1i}\}|\,, \\ C_{2k}^{<} &:= |\{i|k = K_{2i} < K_{1i}\}|\,, \\ C_{k}^{=} &:= |\{i|k = K_{1i} = K_{2i}\}|\,. \end{aligned} \tag{69}$$

These $5m$ values are sufficient for estimating λ_A, λ_B, and λ_X. Actually, the set of these constants can be further reduced, because $C_{10}^{>} = C_{20}^{>} = C_{1(q+1)}^{<} = C_{2(q+1)}^{<} = 0$ always holds.

The log-likelihood function (68) does not always have a strict global maximum point. For example, if all register values of the first HyperLogLog sketch are larger than the corresponding values in the second HyperLogLog sketch, $C_{1k}^{<} = C_{k}^{=} = C_{2k}^{>} = 0$ for all k, 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 maximum likelihood cardinality estimates for the first and second HyperLogLog sketch, respectively. This means that the maximum-likelihood makes no clear statement about the intersection size in this case. The estimate for λ_x could be anything between 0 and $\hat{\lambda}_2$. For comparison, the inclusion-exclusion approach would give $\hat{\lambda}_2$ as estimate. This result is questionable, because there is no evidence at all, that the sets S_1 and S_2 really have common elements.

If it is given that the two HyperLogLog sketches represent disjoint sets, we know in advance that $\lambda_x = 0$. In this case (68) can be splitted into the sum of two unary functions with parameters λ_a and λ_b, respectively. As expected, these two functions are equivalent to the individual log-likelihood functions (37) of both HyperLogLog sketches.

5.2. Computation of the maximum likelihood estimates

The maximum likelihood estimates can be obtained by maximizing (68). Since the three parameters are all non-negative, this is a constrained optimization problem. In order to get rid of these constraints, we use the transformation $\lambda = me^{\varphi}$. This mapping has also the nice property that relative accuracy limits are translated into absolute ones, because $\Delta\varphi = \Delta\lambda/\lambda$. Many optimization algorithm implementations allow the definition of absolute limits rather than relative ones.

The transformed log-likelihood function can be written as

$$
\begin{aligned}
&f(\varphi_a,\varphi_b,\varphi_x) := \log \mathcal{L}(me^{\varphi_a}, me^{\varphi_b}, me^{\varphi_x}|\boldsymbol{K}_1,\boldsymbol{K}_2) = \\
&= -\sum_{k=0}^{q}\left(C_{1k}^{<}+C_{k}^{=}+C_{1k}^{>}\right)\frac{e^{\varphi_a}}{2^k}+\left(C_{2k}^{<}+C_{k}^{=}+C_{2k}^{>}\right)\frac{e^{\varphi_b}}{2^k}+\left(C_{1k}^{<}+C_{k}^{=}+C_{2k}^{<}\right)\frac{e^{\varphi_x}}{2^k} \\
&\quad+\sum_{k=1}^{q} C_{1k}^{<}\log\left(1-e^{-\frac{e^{\varphi_a}+e^{\varphi_x}}{2^k}}\right)+C_{2k}^{<}\log\left(1-e^{-\frac{e^{\varphi_b}+e^{\varphi_x}}{2^k}}\right) \\
&\quad+\sum_{k=1}^{q+1} C_{1k}^{>}\log\left(1-e^{-\frac{e^{\varphi_a}}{2^{\min(k,q)}}}\right)+C_{2k}^{>}\log\left(1-e^{-\frac{e^{\varphi_b}}{2^{\min(k,q)}}}\right) \\
&\quad+\sum_{k=1}^{q+1} C_{k}^{=}\log\left(1-e^{-\frac{e^{\varphi_a}+e^{\varphi_x}}{2^{\min(k,q)}}}-e^{-\frac{e^{\varphi_b}+e^{\varphi_x}}{2^{\min(k,q)}}}+e^{-\frac{e^{\varphi_a}+e^{\varphi_b}+e^{\varphi_x}}{2^{\min(k,q)}}}\right).
\end{aligned}
\tag{70}
$$

Quasi-Newton methods are commonly used for finding the maximum. They require the calculation of the first derivatives which are given by

$$
\begin{aligned}
\frac{\partial f}{\partial \varphi_a} &= -\sum_{k=0}^{q}\left(C_{1k}^{<}+C_{k}^{=}+C_{1k}^{>}\right)\frac{e^{\varphi_a}}{2^k}+\sum_{k=1}^{q} C_{1k}^{<} v\left(\frac{e^{\varphi_a}}{2^k},\frac{e^{\varphi_x}}{2^k}\right)+\sum_{k=1}^{q+1} C_{1k}^{>} v\left(\frac{e^{\varphi_a}}{2^{\min(k,q)}},0\right) \\
&\quad+\sum_{k=1}^{q+1} C_{k}^{=} w\left(\frac{e^{\varphi_a}}{2^{\min(k,q)}},\frac{e^{\varphi_b}}{2^{\min(k,q)}},\frac{e^{\varphi_x}}{2^{\min(k,q)}}\right)
\end{aligned}
\tag{71}
$$

$$
\begin{aligned}
\frac{\partial f}{\partial \varphi_b} &= -\sum_{k=0}^{q}\left(C_{2k}^{<}+C_{k}^{=}+C_{2k}^{>}\right)\frac{e^{\varphi_b}}{2^k}+\sum_{k=1}^{q} C_{2k}^{<} v\left(\frac{e^{\varphi_b}}{2^k},\frac{e^{\varphi_x}}{2^k}\right)+\sum_{k=1}^{q+1} C_{2k}^{>} v\left(\frac{e^{\varphi_b}}{2^{\min(k,q)}},0\right) \\
&\quad+\sum_{k=1}^{q+1} C_{k}^{=} w\left(\frac{e^{\varphi_b}}{2^{\min(k,q)}},\frac{e^{\varphi_a}}{2^{\min(k,q)}},\frac{e^{\varphi_x}}{2^{\min(k,q)}}\right)
\end{aligned}
\tag{72}
$$

$$
\begin{aligned}
\frac{\partial f}{\partial \varphi_x} &= -\sum_{k=0}^{q}\left(C_{1k}^{<}+C_{k}^{=}+C_{2k}^{<}\right)\frac{e^{\varphi_b}}{2^k}+\sum_{k=1}^{q} C_{1k}^{<} v\left(\frac{e^{\varphi_x}}{2^k},\frac{e^{\varphi_a}}{2^k}\right)+\sum_{k=1}^{q} C_{2k}^{<} v\left(\frac{e^{\varphi_x}}{2^k},\frac{e^{\varphi_b}}{2^k}\right) \\
&\quad+\sum_{k=1}^{q+1} C_{k}^{=} u\left(\frac{e^{\varphi_a}}{2^{\min(k,q)}},\frac{e^{\varphi_b}}{2^{\min(k,q)}},\frac{e^{\varphi_x}}{2^{\min(k,q)}}\right)
\end{aligned}
\tag{73}
$$

$$
v(x_1,x_2) := \frac{x_1}{e^{x_1+x_2}-1} = \frac{x_1}{x_1+x_2}\frac{x_1+x_2}{e^{x_1+x_2}-1} \tag{74}
$$

$$
w(x_1,x_2,x_3) := \frac{x_1\left(e^{x_2}-1\right)}{e^{x_1+x_2+x_3}-e^{x_1}-e^{x_2}+1} \tag{75}
$$

$$
u(x_1,x_2,x_3) := \frac{\left(e^{x_1}+e^{x_2}-1\right)x_3}{e^{x_1+x_2+x_3}-e^{x_1}-e^{x_2}+1} \tag{76}
$$

5.3. 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 (55) we define $h_1 := h(2x)$ and $h_2 := h(4x)$. The recursion formula simplifies to

$$h_2 = \frac{x + h_1(1 - h_1)}{x + (1 - h_1)}. \tag{77}$$

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

$$\tilde{h}_2 = \frac{x + \tilde{h}_1(1 - \tilde{h}_1)}{x + (1 - \tilde{h}_1)} \tag{78}$$

The corresponding relative error ε_2 is given by

$$\varepsilon_2 = \frac{\tilde{h}_2}{h_2} - 1. \tag{79}$$

Combination of (77), (78), and (79) yields for its absolute value

$$|\varepsilon_2| = |\varepsilon_1| \, \frac{\left| \frac{h_1(1-2h_1)}{x+h_1(1-h_1)} + \frac{h_1}{x+1-h_1} - \varepsilon_1 \frac{h_1^2}{x+h_1(1-h_1)} \right|}{\left| 1 - \varepsilon_1 \frac{h_1}{x+1-h_1} \right|}. \tag{80}$$

The triangle inequality leads to

$$|\varepsilon_2| \leq |\varepsilon_1| \, \frac{\left| \frac{h_1(1-2h_1)}{x+h_1(1-h_1)} + \frac{h_1}{x+1-h_1} \right| + |\varepsilon_1| \, \frac{h_1^2}{x+h_1(1-h_1)}}{\left| 1 - \varepsilon_1 \frac{h_1}{x+1-h_1} \right|} \tag{81}$$

By numerical means it is easy to show that the inequalities $\left| \frac{h_1(1-2h_1)}{x+h_1(1-h_1)} + \frac{h_1}{x+1-h_1} \right| \leq 0.517$, $\frac{h_1^2}{x+h_1(1-h_1)} \leq 0.436$, and $\frac{h_1}{x+1-h_1} \leq 0.529$ hold for all $x \geq 0$. Therefore, if we additionally assume, for example, $|\varepsilon_1| \leq 0.1$, we get

$$|\varepsilon_2| \leq |\varepsilon_1| \, \frac{0.517 + 0.1 \cdot 0.436}{1 - 0.1 \cdot 0.529} \leq |\varepsilon_1| \cdot 0.592, \tag{82}$$

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 (41) the exact estimate $\hat{x}$ fulfills

$$\hat{x}\sum_{k=0}^{q}\frac{C_k}{2^k}+\sum_{k=1}^{q}C_k h\left(\frac{\hat{x}}{2^k}\right)+C_{q+1}h\left(\frac{\hat{x}}{2^q}\right)=m-C_0 \tag{83}$$

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

$$\left|\tilde{h}(x)-h(x)\right|\leq\varepsilon_h h(x) \tag{84}$$

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

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{C_k}{2^k}+\sum_{k=1}^{q}C_k\tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right)+C_{q+1}\tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right)=m-C_0 \tag{85}$$

Due to (84) there exists some $\alpha\in[-\varepsilon_h,\varepsilon_h]$ for which

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{C_k}{2^k}+(1+\alpha)\left(\sum_{k=1}^{q}C_k h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right)+C_{q+1}h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right)\right)=m-C_0 \tag{86}$$

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

$$\begin{aligned}\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{C_k}{2^k}+(1+\alpha)\left(\sum_{k=1}^{q}C_k h\left(\frac{\hat{x}}{2^k}\right)+\frac{C_k}{2^k}\hat{x}\varepsilon_x\beta\right)+\\ +(1+\alpha)\left(C_{q+1}h\left(\frac{\hat{x}}{2^q}\right)+\frac{C_{q+1}}{2^q}\hat{x}\varepsilon_x\beta\right)=m-C_0\end{aligned} \tag{87}$$

Subtracting (83) multiplied by $(1+\alpha)$ from (87) and resolving ε_x gives

$$\varepsilon_x=\alpha\frac{\hat{x}\sum_{k=0}^{q}\frac{C_k}{2^k}-(m-C_0)}{\hat{x}\left(\sum_{k=0}^{q}\frac{C_k}{2^k}+(1+\alpha)\,\beta\left(\sum_{k=1}^{q}\frac{C_k}{2^k}+\frac{C_{q+1}}{2^q}\right)\right)} \tag{88}$$

Using $|\alpha|\leq\varepsilon_h$, $\beta\geq 0$, and (49) the absolute value of the relative error can be bounded by

$$|\varepsilon_x|\leq|\varepsilon_h|\,\frac{(m-C_0)-\hat{x}\sum_{k=0}^{q}\frac{C_k}{2^k}}{\hat{x}\sum_{k=0}^{q}\frac{C_k}{2^k}} \tag{89}$$

Furthermore, using (46) we finally get

$$|\varepsilon_x|\leq|\varepsilon_h|\,\frac{\frac{1}{2}\sum_{k=1}^{q}\frac{C_k}{2^k}+\frac{C_{q+1}}{2^q}}{\sum_{k=0}^{q}\frac{C_k}{2^k}}\leq|\varepsilon_h|\left(\frac{1}{2}+\frac{\frac{C_{q+1}}{2^q}}{\sum_{k=0}^{q}\frac{C_k}{2^k}}\right)\leq|\varepsilon_h|\left(\frac{1}{2}+\frac{C_{q+1}}{m-C_{q+1}}\right) \tag{90}$$

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] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182 – 209, 1985.

[13] 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.

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

[15] 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.

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

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