New cardinality estimation algorithms for HyperLogLog sketches

Draft version

Otmar Ertl

May 26, 2016

This paper presents new methods to estimate the cardinalities of multisets recorded by HyperLogLog sketches. A theoretically founded extension to the original estimate 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 the cardinality estimates of set intersections when 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 best 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 is chosen to be a power of 2: $m = 2^p$. p is the precision that directly influences the relative error scaling like $1/\sqrt{m}$. The number of bits per register defines the value range that can be represented. Initially all registers are set equal to 0.

2.1. Data element insertion

In order to insert a data element into a HyperLogLog data structure a hash value is calculated. The first p bits of the hash value are used to select one of the 2^p registers. For another q bits, the position of the first 1-bit is determined. If this position 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 number of bits of the consumed hash value $p + q$ limit 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, which will reduce estimation accuracy drastically.

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.

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

2.2. Joint probability distribution of register values

Under the assumption of an ideal 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

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

Algorithm 1 Insertion of a data element D into a HyperLogLog data structure that consists of $m = 2^p$ registers $\vec{k} = (k_1, \ldots, k_m)$ initialized to 0 that can take all integers in $[0, q+1]$.

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

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

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

Note that 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 also contains all the information about the HyperLogLog sketch that is required for cardinality estimation: $q = \dim(\vec{c}) - 2$ and $p = \log_2 \|\vec{c}\|_1$.

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 register values are distributed according to

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

This factorization shows that all register values are now 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 given by

$$P(K \leq k|\lambda) = \begin{cases} 0 & k < 0 \\ e^{-\frac{\lambda}{m 2^k}} & 0 \leq k \leq q \\ 1 & k > q \end{cases} \tag{5}$$

Due to the simpler probability mass function, it is easier to find an estimate $\hat{\lambda}$ for the Poisson rate λ than for the cardinality n in the fixed-size model (1). Since, the expectation of the cardinality under the Poisson model is equal to λ, it makes sense to take $\hat{\lambda}$ direclty as estimate for the cardinality, hence, $\hat{n} := \hat{\lambda}$. The Poisson approximation is known to work well for large rates. However, as simulation results show later, the Poisson approximation gives high quality estimates for small cardinalities as well.

3. Original cardinality estimation method

The original cardinality estimate [6] is based on the idea that approximately $m2^{k_j}$ distinct element insertions are needed until the value of register j reaches k_j. Given that, a 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 [13]. Finally, the harmonic mean was found to give even better estimates as it is inherently less sensitive to outliers. It is used for

the *raw* estimate of the HyperLogLog algorithm and is given by

$$\hat{n}_{\text{raw}} = \frac{\alpha_m m^2}{\sum_{j=1}^{m} 2^{-k_j}} = \frac{\alpha_m m^2}{\sum_{k=0}^{q+1} c_k 2^{-k}} \tag{6}$$

where α_m is a correction factor that is fixed for a given number of registers m and which was derived to be [6]

$$\alpha_m := \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1} \tag{7}$$

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 $p \geq 20$. For HyperLogLog sketches that are used in practice with more than 256 registers ($p \geq 8$), it is sufficient to use the limit $\alpha_\infty := \lim_{m\to\infty} \alpha_m$ as approximation

$$\alpha_m \approx \alpha_\infty = \frac{1}{2 \log 2} \approx 0.7213475, \tag{8}$$

because the corresponding approximation error is already small compared to the estimation error.

Figure 1 shows the distribution of the relative error of the *raw* estimate for 10 000 randomly generated HyperLogLog sketches over the number of distinct element insertions. Obviously, the *raw* estimate is biased for small and large cardinalities and fails to return accurate estimates. As a consequence, to cover the entire range of cardinalities, a small and a large correction have been proposed, respectively.

For small cardinalities the same estimator as used by the linear probabilistic counting algorithm [12] which only incorporates the number of registers equal to 0 denoted by c_0:

$$\hat{n}_{\text{small}} = m \log \left(\frac{m}{c_0} \right) \tag{9}$$

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

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* estimate 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

$$\hat{n}_{\text{large}} = -2^{32} \log \left(1 - \frac{\hat{n}_{\text{raw}}}{2^{32}} \right). \tag{10}$$

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 procedure for estimating the cardinality from a HyperLogLog data structure using 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 (6)
  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 (9)
    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 (10)
  end if
end function

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. The empirically determined bias correction data can be either stored as set of interpolation points [7], as lookup table [8], or as best-fitting polynomial [9].

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. A simple approach to avoid the large range correction at all is the use of larger hash values ($p + q > 32$) which shift the biased region of $\hat{n}_{\text{raw}}$ to higher cardinalities. However, in case $q \geq 30$ 6-bit registers as used in [7] are required to allow representing all values in the range $[0, q+1]$. If 64-bit hash values are used as proposed in [7] ($p + q = 64$), a correction for large cardinalities can be circumvented. Cardinalities in the order of 2^{64} for which saturated register become a problem never occur in practice.

3.1. Deriviation of the *raw* estimate

Assume following integer distribution

$$P(K \leq k|\lambda) = e^{-\frac{\lambda}{m2^k}}. \tag{11}$$

Note that, this distribution differs from the register value distribution under the Poisson model (5), whose support is limited to the range $[0, q + 1]$. The expectation of 2^{-k} is

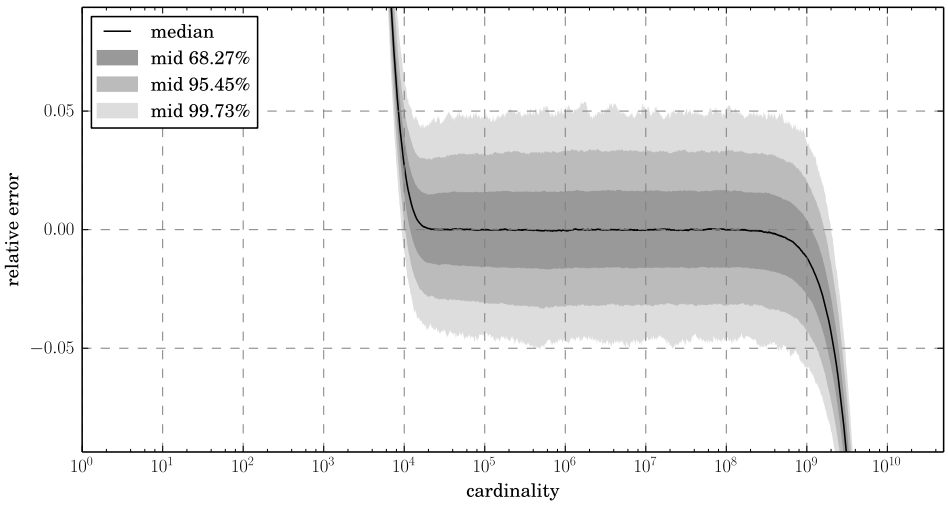

Figure 1: The distribution of the relative estimation error over the cardinality as obtained by the raw estimation method [6]. 10 000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$ have been evaluated.

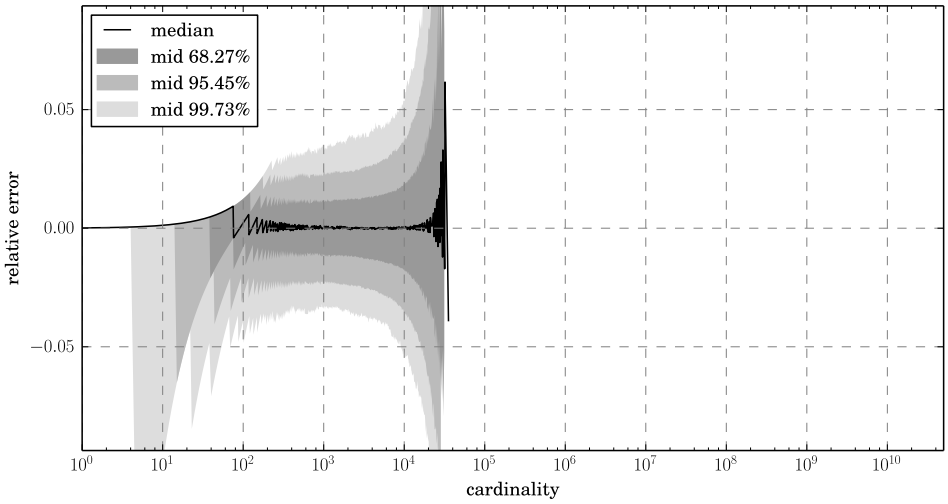

Figure 2: The distribution of the relative estimation error for the small range correction.

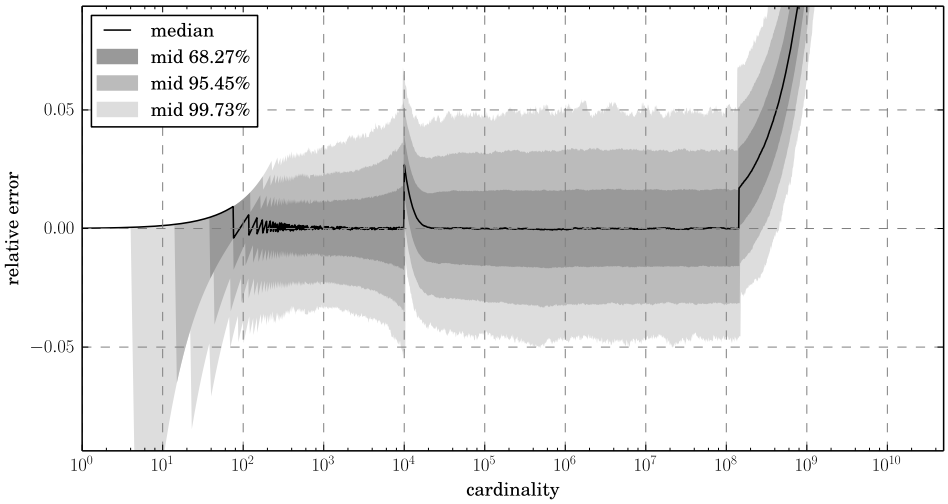

Figure 3: The distribution of the relative estimation error over the cardinality as obtained by the original estimation method [6]. 10 000 randomly generated HyperLogLog data structures with parameters $p = 12$ and $q = 20$ have been evaluated.

given by

$$\mathbb{E}(2^{-k}) = \sum_{k=-\infty}^{\infty} 2^{-k} e^{-\frac{\lambda}{m}2^{-k}} \left(1 - e^{-\frac{\lambda}{m}2^{-k}}\right) = \frac{1}{2} \sum_{k=-\infty}^{\infty} 2^{k} e^{-\frac{\lambda}{m}2^{k}} = \frac{m\,\xi(\log_2(\lambda/m))}{2\log(2)\lambda} \quad (12)$$

where the function

$$\xi(y) := \log(2) \sum_{k=-\infty}^{\infty} 2^{k+y} e^{-2^{k+y}} \quad (13)$$

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

Let $k_1, \ldots, k_m$ be a sample taken from the distribution described by (11). For large sample sizes $m \to \infty$ we have asymptotically

$$\mathbb{E}\left(\frac{1}{2^{-k_1} + \ldots + 2^{-k_m}}\right) \underset{m\to\infty}{=} \frac{1}{\mathbb{E}\left(2^{-k_1} + \ldots + 2^{-k_m}\right)} = \frac{1}{m\mathbb{E}\left(2^{-k}\right)}. \quad (14)$$

Together with (14) we get

$$\lambda = \mathbb{E}\left(\frac{m^2 \xi(\log_2(\lambda/m))}{2\log(2)\left(2^{-k_1} + \ldots + 2^{-k_m}\right)}\right) \quad \text{for } m \to \infty \quad (15)$$

and therefore, the estimator

$$\hat{\lambda} = \frac{m^2}{2\log(2)\left(2^{-k_1} + \ldots + 2^{-k_m}\right)} \quad (16)$$

for the Poisson parameter, is asymptotically almost unbiased with maximum relative offset ε_ξ. This estimator corresponds to the *raw* estimate, already presented in (6) and can also be written as

$$\hat{\lambda} = \frac{m^2}{2\log(2)\sum_{k=-\infty}^{\infty} c'_k 2^{-k}} \tag{17}$$

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

3.2. Limitations of the *raw* estimate

As we have already seen in Figure 1, the *raw* estimate does not work very well for small or large cardinalities. The reason for this is that the distribution of register values (5) is different from (11) for which the *raw* estimate was derived. Since a random variable K' that obeys (11) can be transformed into random variable K that follows (5) using the transformation $K = \min(\max(K', 0), q+1)$, the register values of a HyperLogLog sketch can be seen as the result after applying this transformation to a sample distributed according to (11). The corresponding multiplicities are related by

$$\begin{aligned} c_0 &= \textstyle\sum_{k=-\infty}^{0} c'_k, \\ c_k &= c'_k, \quad \text{for } k \in [1, q], \\ c_{q+1} &= \textstyle\sum_{k=q+1}^{\infty} c'_k. \end{aligned} \tag{18}$$

Obviously, as long as $c_0 \approx c'_0$ and $c'_{q+1} \approx c_{q+1}$ the set of register values is similar to the originating sample. In case $c_0 = 0$ and $c_{q+1} = 0$, we can even be sure that the register values are identical to the originating sample. However, for small amd large cardinalities where c_0 and c_{q+1} approach m, respectively, the *raw* estimate needs to be corrected.

3.3. Correcting the *raw* estimate

Given a HyperLogLog sketch with its multiplicity vector $\vec{c}$, we try to find estimates $\hat{c}'_k$ for c'_k for all $k \in \mathbb{Z}$ and use them instead in the estimation formula (17). For $k \in [1, q]$ where $c_k = c'_k$ we have the trivial estimators

$$\hat{c}'_k := c_k \quad \text{for } k \in [1, q]. \tag{19}$$

To get estimators for all other k, we consider their expectations

$$\mathbb{E}(c'_k) = m e^{-\frac{\lambda}{m2^k}} \left(1 - e^{-\frac{\lambda}{m2^k}}\right). \tag{20}$$

From (5) we know that $\mathbb{E}\left(\frac{c_0}{m}\right) = e^{-\frac{\lambda}{m}}$ and $\mathbb{E}\left(1 - \frac{c_{q+1}}{m}\right) = e^{-\frac{\lambda}{m2^q}}$, and therefore, we can also write

$$\mathbb{E}(c'_k) = m\mathbb{E}\left(\frac{c_0}{m}\right)^{2^{-k}} \left(1 - \mathbb{E}\left(\frac{c_0}{m}\right)^{2^{-k}}\right) \tag{21}$$

and

$$\mathbb{E}(c'_k) = m\mathbb{E}\left(1 - \frac{c_{q+1}}{m}\right)^{2^{q-k}} \left(1 - \mathbb{E}\left(1 - \frac{c_{q+1}}{m}\right)^{2^{q-k}}\right), \tag{22}$$

which motivates us to use

$$\hat{c}'_k := m \cdot \left(\frac{c_0}{m}\right)^{2^{-k}} \cdot \left(1 - \left(\frac{c_0}{m}\right)^{2^{-k}}\right) \tag{23}$$

as estimator for $k \leq 0$ and

$$\hat{c}'_k := m \cdot \left(1 - \frac{c_{q+1}}{m}\right)^{2^{q-k}} \cdot \left(1 - \left(1 - \frac{c_{q+1}}{m}\right)^{2^{q-k}}\right) \tag{24}$$

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

Inserting all these estimators into (17) gives

$$\hat{\lambda} = \frac{m^2}{2\log(2)\sum_{k=-\infty}^{\infty} \hat{c}'_k 2^{-k}} = \frac{m^2}{2\log(2)\left(\sigma(c_0, m) + \sum_{k=1}^{q} c_k 2^{-k} + \tau(c_{q+1}, m)2^{-(q+1)}\right)} \tag{25}$$

where $\sigma(c_0)$ and $\tau(c_{q+1})$ replace c_0 and c_{q+1} in the *raw* estimate (6), respectively. The functions σ and τ are given by

$$\sigma(c, m) := m \cdot \left(\frac{c}{m} + \sum_{k=1}^{\infty} \left(\frac{c}{m}\right)^{2^k} 2^{k-1}\right) \tag{26}$$

and

$$\tau(c, m) := m \cdot \sum_{k=1}^{\infty} \left(1 - \frac{c_{q+1}}{m}\right)^{2^{-k}} \cdot \left(1 - \left(1 - \frac{c_{q+1}}{m}\right)^{2^{-k}}\right) \cdot 2^{-(k-1)} \tag{27}$$

For the special case of linear probabilistic counting with $q = 0$, we get for the estimator (25)

$$\hat{\lambda} = \frac{m}{\log(2)\sum_{k=-\infty}^{\infty} \left(\frac{c_0}{m}\right)^{2^k} 2^k} = \frac{m \log\left(\frac{m}{c_0}\right)}{\xi\left(\log_2\left(\log\left(\frac{m}{c_0}\right)\right)\right)} \approx m \log\left(\frac{m}{c_0}\right). \tag{28}$$

Here we used function ξ defined in (13) which is known to be approximately equal to 1. Hence, for this special case we get as expected almost the same estimator as given in (9).

3.4. Estimation algorithm

see Algorithm 3

3.5. Estimation error

see Figure 4, Figure 5, Figure 6, Figure 7, Figure 8,

3.6. Performance

see Figure 9

Algorithm 3 Cardinality estimation algorithm based on the corrected *raw* estimate.

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

function $\sigma(c, m)$
  if $c = m$ **then**
    return ∞
  end if
  $x \leftarrow c/m$
  $y \leftarrow 1$
  $z \leftarrow x$
  repeat
    $x \leftarrow x \cdot x$
    $z' \leftarrow z$
    $z \leftarrow z + x \cdot y$
    $y \leftarrow 2 \cdot y$
  until $z = z'$
  return $m \cdot z$
end function

function $\tau(c, m)$
  $x \leftarrow (m - c)/m$
  $y \leftarrow 1$
  $z \leftarrow 0$
  repeat
    $x \leftarrow \sqrt{x}$
    $z' \leftarrow z$
    $z \leftarrow z + (1 - x) \cdot x \cdot y$
    $y \leftarrow 0.5 \cdot y$
  until $z = z'$
  return $m \cdot z$
end function

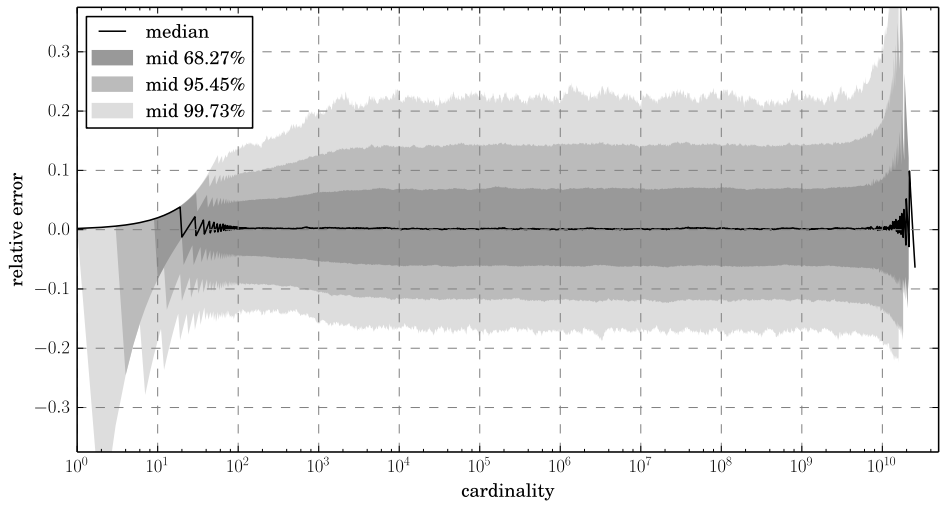

Figure 4: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

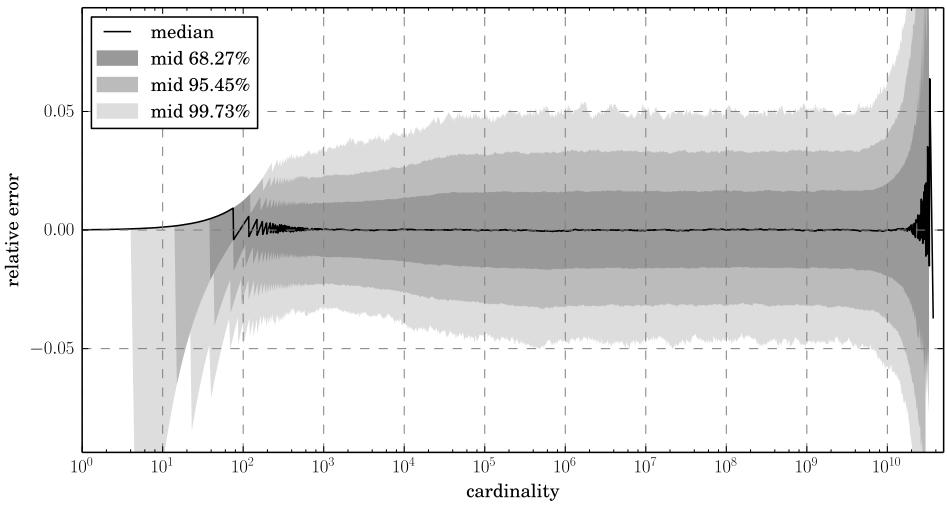

Figure 5: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

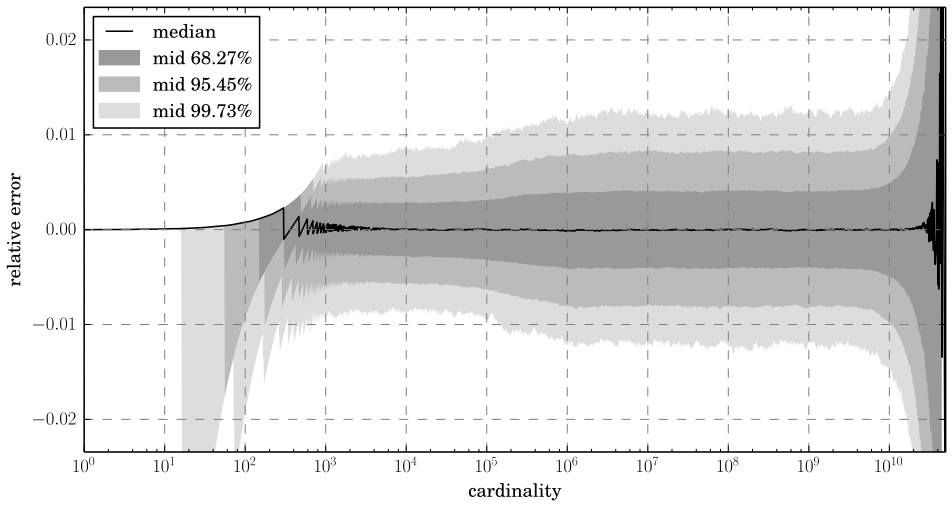

Figure 6: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

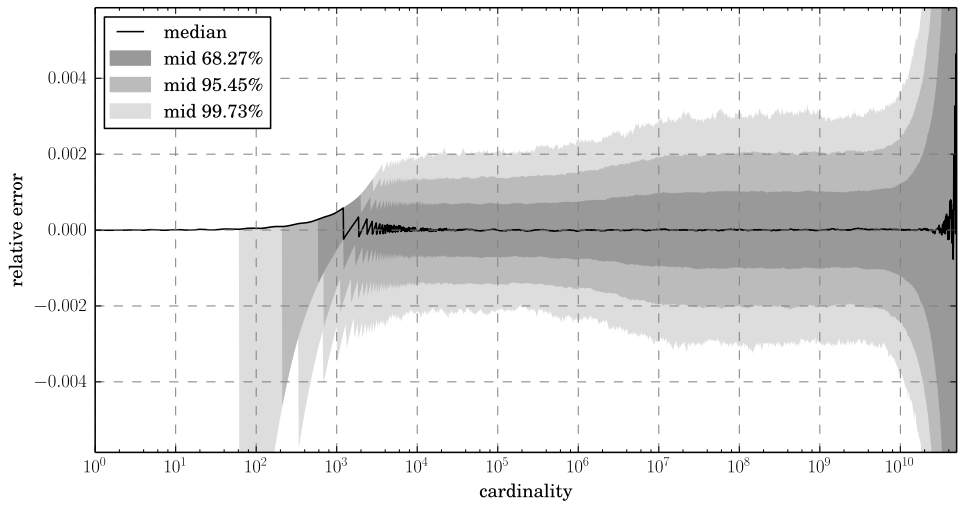

Figure 7: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 20$ and $q = 12$.

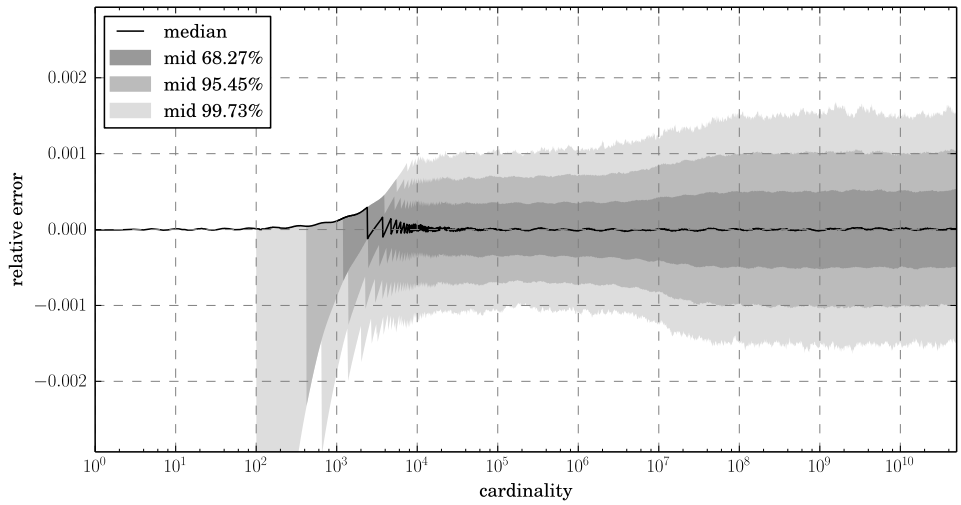

Figure 8: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 42$.

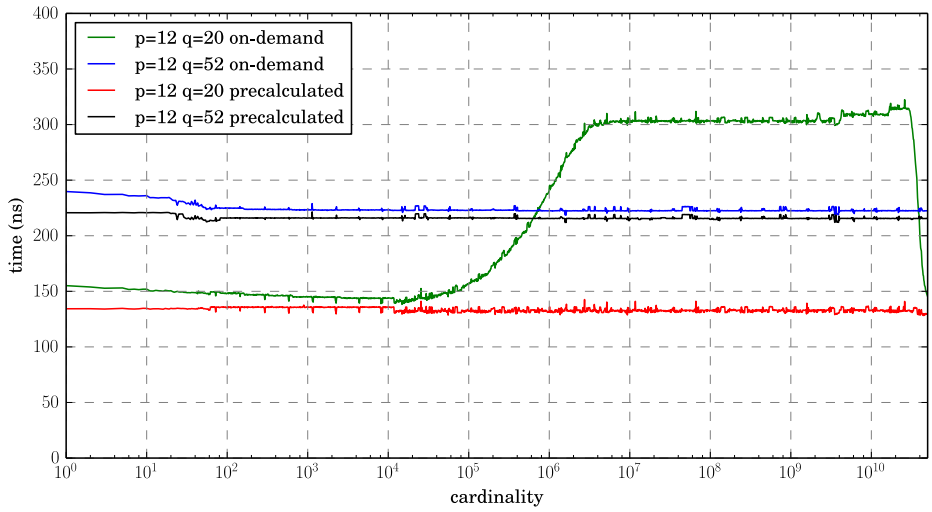

Figure 9: Average computation time as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3GHz when estimating the cardinality from HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively. Both cases, σ and τ precalculated or calculated on-demand have been considered.

4. Maximum likelihood estimation

The maximum likelihood method was already applied to HyperLogLog sketches in [14]. However, there, a different element insertion algorithm was assumed that works with time complexity $\mathcal{O}(m)$, because each register calculates its own hash value from the inserted element and updates its current value accordingly. In comparison, Algorithm 1 uses only a single hash value that defines which register needs to be updated. As a consequence, the register values are statistically dependent, and the Poisson approximation is needed to make the register values statistically independent, which allows the factorization of the joint probability distribution of all register values.

In the following we derive a new robust and efficient cardinality estimation algorithm that is based on the maximum likelihood method. 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, 13].

4.1. Log-likelihood function

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

$$\log \mathcal{L}(\lambda|\vec{k}) = -\frac{\lambda}{m}\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \log\left(1 - e^{-\frac{\lambda}{m2^k}}\right) + c_{q+1}\log\left(1 - e^{-\frac{\lambda}{m2^q}}\right) \tag{29}$$

and

$$\frac{d}{d\lambda}\log \mathcal{L}(\lambda|\vec{k}) = -\frac{1}{\lambda}\left(\frac{\lambda}{m}\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \frac{\frac{\lambda}{m2^k}}{1 - e^{\frac{\lambda}{m2^k}}} + c_{q+1}\frac{\frac{\lambda}{m2^q}}{1 - e^{\frac{\lambda}{m2^q}}}\right). \tag{30}$$

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

$$\hat{\lambda} = m\hat{x}, \tag{31}$$

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

$$f(x) := x\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \frac{\frac{x}{2^k}}{1 - e^{\frac{x}{2^k}}} + c_{q+1}\frac{\frac{x}{2^q}}{1 - e^{\frac{x}{2^q}}}. \tag{32}$$

This function can also be written as

$$f(x) := x\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{x}{2^k}\right) + c_{q+1}h\left(\frac{x}{2^q}\right) - (m - c_0), \tag{33}$$

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

$$h(x) := 1 - \frac{x}{e^x - 1}. \tag{34}$$

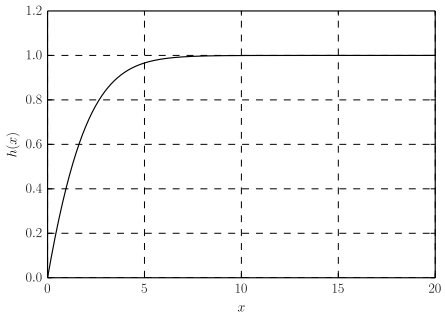

Figure 10: The function $h(x)$.

$h(x)$ is strictly increasing and concave as can be seen in Figure 10. 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. $c_q = m$ corresponds to the case with all registers equal to the maximum value $(q+1)$, for which the maximum likelihood estimate would be infinity.

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 (33) as

$$f(x) := c_0 x + c_1 \left(h\left(\frac{x}{2^1}\right) + \frac{x}{2^1} - 1 \right) + c_2 \left(h\left(\frac{x}{2^2}\right) + \frac{x}{2^2} - 1 \right) + \ldots \\ \cdots + c_q \left(h\left(\frac{x}{2^q}\right) + \frac{x}{2^q} - 1 \right) + c_{q+1} \left(h\left(\frac{x}{2^q}\right) - 1 \right). \quad (35)$$

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 (32) can be solved analytically for the special case $q = 0$ which corresponds to the already mentioned linear probabilistic counting algorithm. In this case, the maximum likelihood method under the Poisson model directly leads to the estimator presented in [12] and which was also used for small range estimation (9). 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 very 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 (33) gives an upper bound for $f(x)$:

$$f(x) \leq x \sum_{k=0}^{q} \frac{c_k}{2^k} + (m - c_0) \cdot h\left(x \cdot \left(\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}\right)\right) - (m - c_0)\,. \tag{36}$$

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

$$\hat{x} \geq \frac{m - c_0}{\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}} \log\left(1 + \frac{\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}{\sum_{k=0}^{q} \frac{c_k}{2^k}}\right). \tag{37}$$

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

$$\hat{x} \geq \frac{m - c_0}{c_0 + \frac{3}{2}\sum_{k=1}^{q} \frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}. \tag{38}$$

Using the monotonicity of h, the lower bound

$$f(x) \geq x \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{x}{2^{k'_{\max}}}\right) + c_{q+1} h\left(\frac{x}{2^{k'_{\max}}}\right) - (m - c_0)\,, \tag{39}$$

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

$$\hat{x} \leq 2^{k'_{\max}} \log\left(1 + \frac{m - c_0}{2^{k'_{\max}} \sum_{k=0}^{q} \frac{c_k}{2^k}}\right) \tag{40}$$

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

$$\hat{x} \leq \frac{m - c_0}{\sum_{k=0}^{q} \frac{c_k}{2^k}}. \tag{41}$$

Note, if the HyperLogLog sketch is in the intermediate range, where $c_0 = c_{q+1} = 0$ the bounds (38) and (41) differ only by a constant factor from the *raw* estimate (6). Hence, the maximum likelihood method leads directly to the harmonic mean that is used for the *raw* estimate. If $c_0 > 0$ or $c_{q+1} > 0$ the bounds do no longer follow the harmonic mean and this is also the reason why the *raw* estimate fails for small and large cardinalities.

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

$$x_i = x_{i-1} - (x_{i-1} - x_{i-2}) \frac{f(x_{i-1})}{f(x_{i-1}) - f(x_{i-2})} \tag{42}$$

If $x_0 = 0$ with $f(x_0) = -(m - c_0)$, and x_1 is equal to one of the derived lower bounds (37) or (38), the sequence $\{x_i\}$ is montone increasing. Using the definitions

$$\Delta x_i := x_i - x_{i-1} \tag{43}$$

and

$$g(x) := f(x) + (m - c_0) = x \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{x}{2^k}\right) + c_{q+1} h\left(\frac{x}{2^q}\right) \tag{44}$$

the iteration scheme can also be written as

$$\Delta x_i = \Delta x_{i-1} \frac{(m - c_0) - g(x_{i-1})}{g(x_{i-1}) - g(x_{i-2})} \tag{45}$$

$$x_i = x_{i-1} + \Delta x_i \tag{46}$$

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 following results we have used $\varepsilon = 10^{-2}$.

4.4. The new cardinality estimation algorithm

In order to get a fast cardinality estimation algorithm, it is crucial to minimize the evaluation costs for (44). 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 $\{\frac{x}{2^{k'_{\max}}}, \frac{x}{2^{k'_{\max}-1}} \ldots, \frac{x}{2^{k_{\min}}}\}$ where $k'_{\max} := \min(k_{\max}, q)$. This series corresponds to a geometric series with ratio 2. A straightforward calculation using (34) 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

$$h\left(2x\right) = \frac{x + 2h\left(x\right)\left(1 - h\left(x\right)\right)}{x + 2\left(1 - h\left(x\right)\right)} \tag{47}$$

or

$$h\left(\frac{x}{2^k}\right) = \frac{\frac{x}{2^{k+2}} + h\left(\frac{x}{2^{k+1}}\right)\left(1 - h\left(\frac{x}{2^{k+1}}\right)\right)}{\frac{x}{2^{k+2}} + \left(1 - h\left(\frac{x}{2^{k+1}}\right)\right)} \tag{48}$$

Note, this recursive formula is stable in a sense that the relative error of $h\left(2x\right)$ is smaller than that of $h\left(x\right)$ 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 $x = 0$

$$h(x) = \frac{x}{2} - \frac{x^2}{12} + \frac{x^4}{720} - \frac{x^6}{30240} + \mathcal{O}(x^8) \tag{49}$$

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

$$h(x) = x' - x''/3 + \left(x''x''\right)\left(1/45 - x''/472.5\right) + \mathcal{O}(x^8) \tag{50}$$

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 (40) 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 approximation. $h\left(\frac{x}{2^{k'_{\max}}}\right)$ is finally obtained after $e + 1 - k'_{\max}$ iterations using (48).

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

4.5. Estimation error

In order to verify the new estimation algorithm, we generated 10 000 HyperLogLog sketches and inserted up to 50 billion unique elements. Element hash values have been 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 11 shows the distribution of the relative error of the estimated cardinality using Algorithm 4 compared to the true cardinality for $p = 12$ and $q = 20$. As the median shows, the error is essentially unbiased over the entire cardinality range except for very small values. 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* estimate (Figure 1) or the original method (Figure 3).

Algorithm 4 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 (38)
  else
    $x \leftarrow m'/b \cdot \log(1 + b/a)$ ▷ strong lower bound (37)
  end if

Algorithm 4 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 (50)
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 (48), at this point $x' = \frac{x}{2^{k+2}}$
$x' \leftarrow 2x'$
end for
$g \leftarrow c' \cdot h$ ▷ compare (44)
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 (48), 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 (45)
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

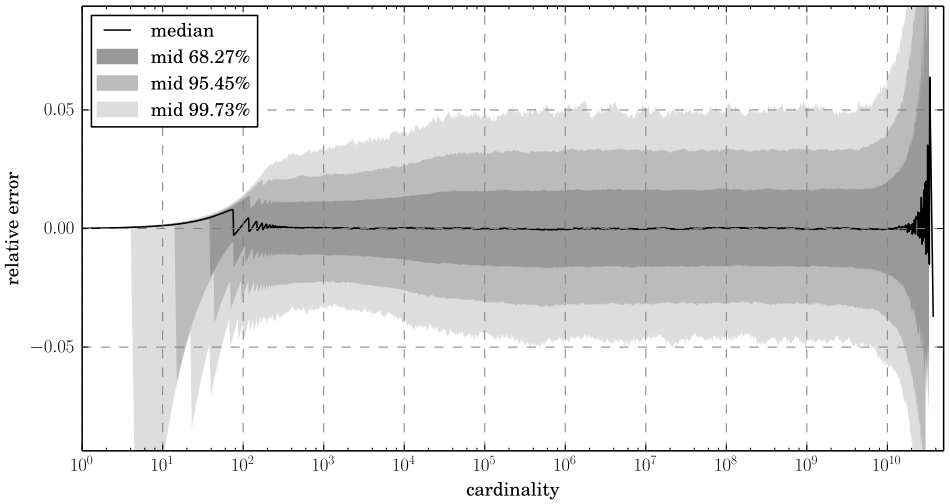

Figure 11: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 20$.

The new estimation algorithm also works well for other HyperLogLog configurations. First we considered the original HyperLogLog algorithm that used a 32-bit hash function ($p+q = 32$). The relative estimation error for precisions $p = 8$, $p = 16$, $p = 20$ are shown in Figure 12, Figure 13, and Figure 14, 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 inherently unbiased estimates.

As proposed in [7], the operating range can be extended by replacing the 32-bit hash function by a 64-bit hash function. Figure 15 shows the relative error for such a HyperLogLog 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 15 shows, when compared to the 32-bit hash value case given in Figure 11, 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 16, which could be enough for many applications.

TODO Figure 17

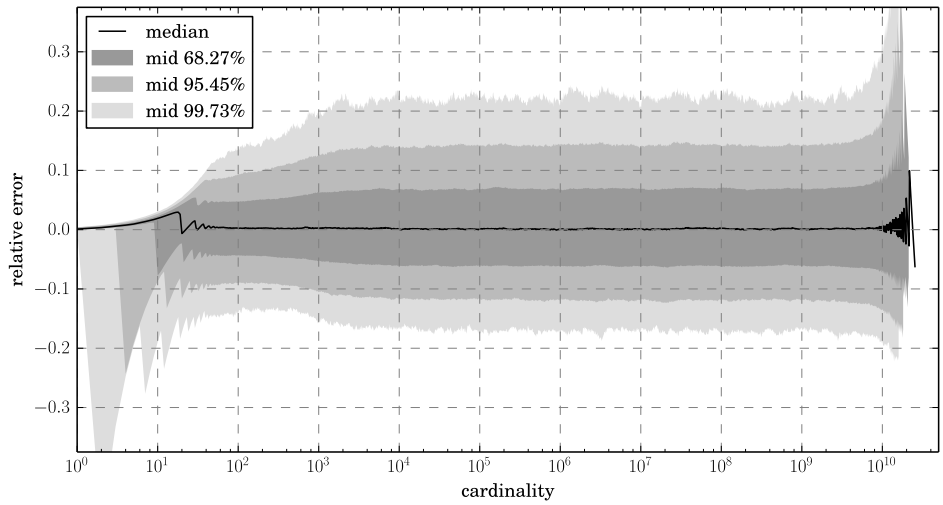

Figure 12: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 8$ and $q = 24$.

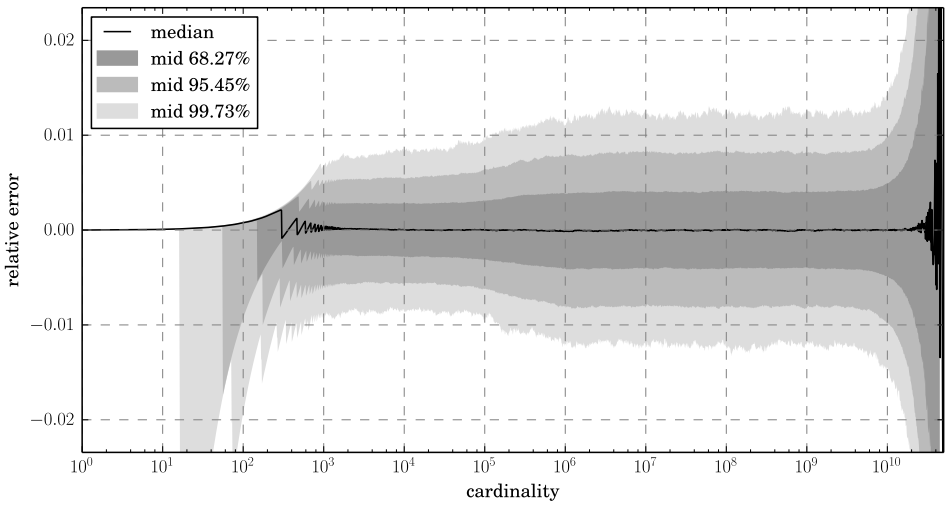

Figure 13: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 16$ and $q = 16$.

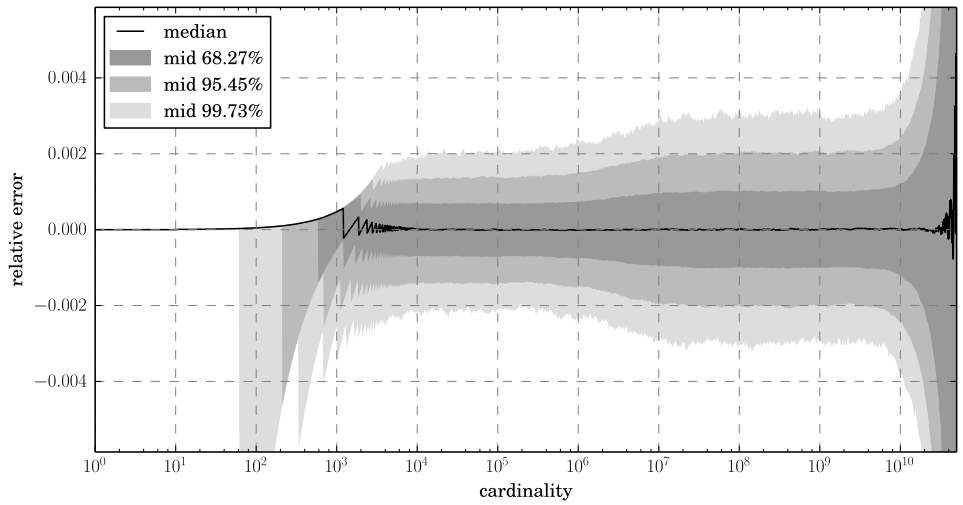

Figure 14: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 20$ and $q = 12$.

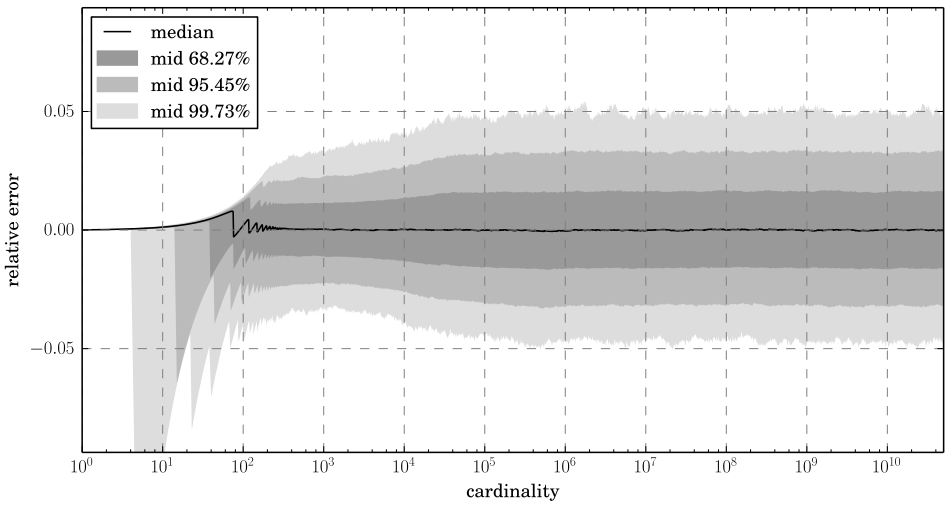

Figure 15: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 52$.

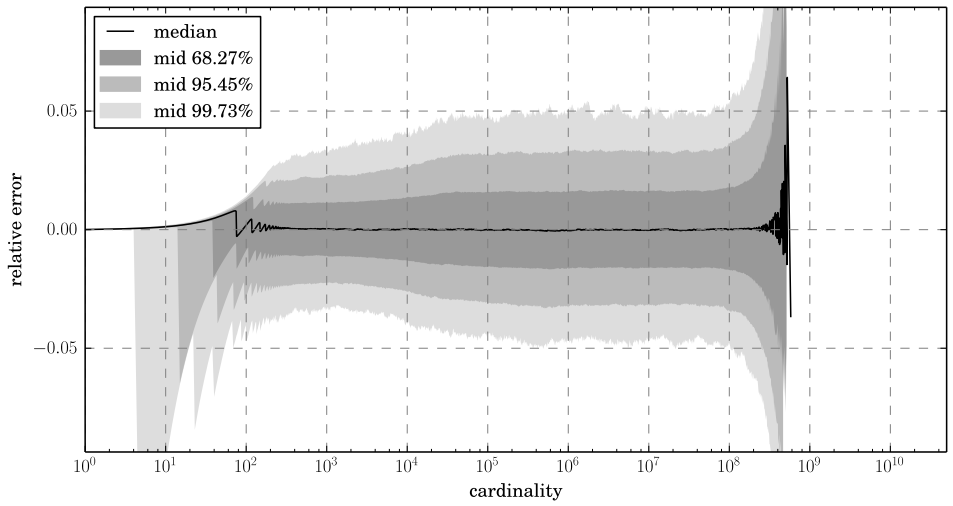

Figure 16: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 12$ and $q = 14$.

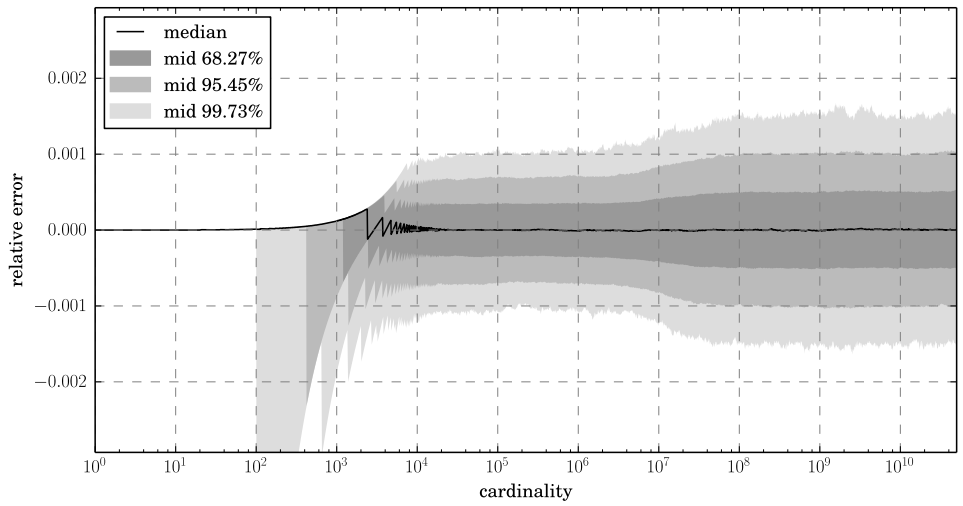

Figure 17: Relative estimation error as a function of the true cardinality for a HyperLogLog sketch with parameters $p = 22$ and $q = 42$.

4.6. Performance

In order to prove that the new algorithm is not only accurate, but also fast, we investigated the average computation time for estimating the cardinality from a given HyperLogLog sketch. For different cardinalities we loaded the multiplicity vectors of 1000 precalculated and 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 new estimation algorithm. For each evaluated cardinality value the average execution time was calculated after 100 cycles which corresponds to 100 000 algorithm executions. The results for HyperLogLog configurations $p = 12, q = 20$ and $p = 12, q = 52$ are shown in Figure 18. All these benchmarks where carried out on an Intel Core i5-2500K clocking at 3.3GHz. As can be seen, the average calculation time did never exceed 700 ns.

The numbers do not yet include the required processing time to calculate the multiplicity vector 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 processing time numbers are quite satisfying.

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 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,

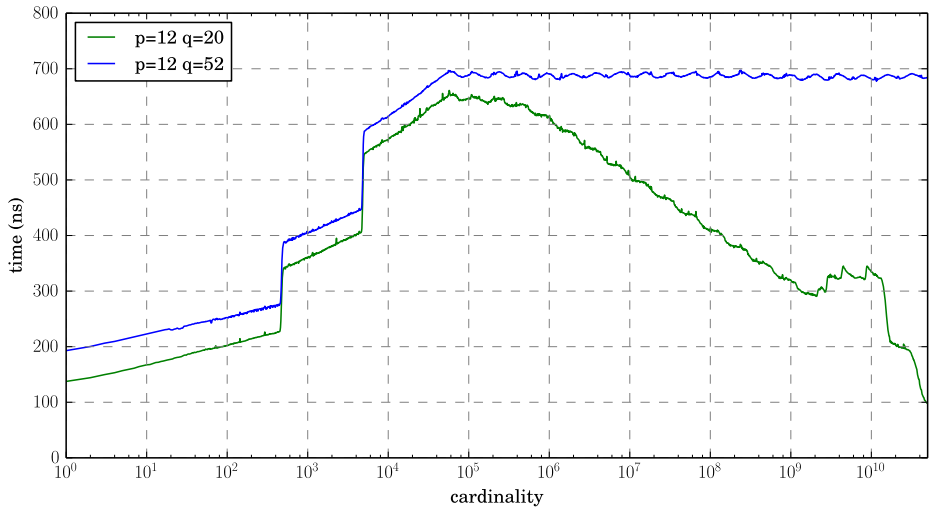

Figure 18: Average computation time as a function of the true cardinality with an Intel Core i5-2500K clocking at 3.3GHz when estimating the cardinality from HyperLogLog sketches with parameters $p = 12$, $q = 20$ and $p = 12$, $q = 52$, respectively.

B, and X, respectively. The corresponding values in sketches for S_1 and S_2 are given by

$$K_1 = \max(K_a, K_x), \quad K_2 = \max(K_b, K_x) \tag{51}$$

Their joint cumulative probability function is given as

$$\begin{aligned} P(K_1 \leq k_1 \wedge K_2 \leq k_2) &= P(\max(K_a, K_x) \leq k_1 \wedge \max(K_b, K_x) \leq k_2) \\ &= P(K_a \leq k_1 \wedge K_b \leq k_2 \wedge K_x \leq \min(k_1, k_2)) \\ &= P(K_a \leq k_1) P(K_b \leq k_2) P(K_x \leq \min(k_1, k_2)) \end{aligned} \tag{52}$$

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 (5). 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

$$P(K_1 \leq k_1 \wedge K_2 \leq k_2) = \begin{cases} 0 & k_1 < 0 \vee k_2 < 0 \\ e^{-\frac{\lambda_a}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}} - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}} & 0 \leq k_1 \leq q \wedge 0 \leq k_2 \leq q \\ e^{-\frac{\lambda_b + \lambda_x}{m2^{k_2}}} & 0 \leq k_2 \leq q < k_1 \\ e^{-\frac{\lambda_a + \lambda_x}{m2^{k_1}}} & 0 \leq k_1 \leq q < k_2 \\ 1 & q < k_1 = k_2 \end{cases} \tag{53}$$

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

$$\begin{aligned}\rho(k_1,k_2) = {} & P(K_1 \leq k_1 \wedge K_2 \leq k_2) - P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2) \\ & - P(K_1 \leq k_1 \wedge K_2 \leq k_2 - 1) + P(K_1 \leq k_1 - 1 \wedge K_2 \leq k_2 - 1)\end{aligned} \tag{54}$$

which finally gives

$$\rho(k_1,k_2) = \begin{cases} e^{-\frac{\lambda_a+\lambda_x}{m} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 0 = k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_b}{m2^{q}}}\right) & 0 = k_1 < k_2 = q+1 \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}} - \frac{\lambda_b}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right)\left(1 - e^{-\frac{\lambda_b}{m2^{k_2}}}\right) & 1 \leq k_1 < k_2 \leq q \\ e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right)\left(1 - e^{-\frac{\lambda_b}{m2^{q}}}\right) & 1 \leq k_1 < k_2 = q+1 \\ e^{-\frac{\lambda_b+\lambda_x}{m} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 0 = k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m}} \left(1 - e^{-\frac{\lambda_a}{m2^{q}}}\right) & 0 = k_2 < k_1 = q+1 \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}} - \frac{\lambda_a}{m2^{k_1}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right)\left(1 - e^{-\frac{\lambda_a}{m2^{k_1}}}\right) & 1 \leq k_2 < k_1 \leq q \\ e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}} \left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right)\left(1 - e^{-\frac{\lambda_a}{m2^{q}}}\right) & 1 \leq k_2 < k_1 = q+1 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m}} & 0 = k_1 = k_2 \\ e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{k}}} \left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{k}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{k}}}\right) & 1 \leq k_1 = k_2 = k \leq q \\ 1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{q}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{q}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{q}}} & k_1 = k_2 = q+1 \end{cases} \tag{55}$$

The logarithm of the joint probability mass function can be written using Iverson bracket notation ([true] := 1, [false] := 0) as

$$\begin{aligned}\log(\rho(k_1,k_2)) = {} & -\frac{\lambda_a}{m2^{k_1}}[k_1 \leq q] - \frac{\lambda_b}{m2^{k_2}}[k_2 \leq q] - \frac{\lambda_x}{m2^{\min(k_1,k_2)}}[k_1 \leq q \vee k_2 \leq q] \\ & + \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{k_1}}}\right)[1 \leq k_1 < k_2] \\ & + \log\left(1 - e^{-\frac{\lambda_a}{m2^{\min(k_1,q)}}}\right)[k_2 < k_1] \\ & + \log\left(1 - e^{-\frac{\lambda_b+\lambda_x}{m2^{k_2}}}\right)[1 \leq k_2 < k_1] \\ & + \log\left(1 - e^{-\frac{\lambda_b}{m2^{\min(k_2,q)}}}\right)[k_1 < k_2] \\ & + \log\left(1 - e^{-\frac{\lambda_a+\lambda_x}{m2^{\min(k_1,q)}}} - e^{-\frac{\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}} + e^{-\frac{\lambda_a+\lambda_b+\lambda_x}{m2^{\min(k_1,q)}}}\right)[1 \leq k_1 = k_2]\end{aligned} \tag{56}$$

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

$$\rho(\vec{k}_1, \vec{k}_2) = \prod_{k_1=0}^{q+1} \prod_{k_2=0}^{q+1} \rho(k_1, k_2)^{c_{k_1 k_2}}. \tag{57}$$

Here we have used the multiplicity matrix $\mathbf{c} = (c_{k_1 k_2})_{k_1 k_2 \in [0, q+1]}$ defined by

$$c_{k_1 k_2} := |\{(i, j) \,|k_{i,1} = k_1 \wedge k_{j,2} = k_2\}| \tag{58}$$

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

$$\log \mathcal{L}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = \sum_{k_1=0}^{q+1} \sum_{k_2=0}^{q+1} c_{k_1 k_2} \log(\rho(k_1, k_2)) \tag{59}$$

Insertion of (56) results in

$$\begin{aligned}
\log \mathcal{L}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{\lambda_a}{m} \sum_{k=0}^{q} \frac{c_k^{\leftarrow} + c_{kk} + c_k^{\rightarrow}}{2^k} - \frac{\lambda_b}{m} \sum_{k=0}^{q} \frac{c_k^{\uparrow} + c_{kk} + c_k^{\downarrow}}{2^k} \\
&- \frac{\lambda_x}{m} \sum_{k=0}^{q} \frac{c_k^{\rightarrow} + c_{kk} + c_k^{\downarrow}}{2^k} \\
&+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a + \lambda_x}{m 2^{\min(k,q)}}}\right) c_k^{\rightarrow} + \log\left(1 - e^{-\frac{\lambda_a}{m 2^{\min(k,q)}}}\right) c_k^{\leftarrow} \\
&+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_b + \lambda_x}{m 2^{\min(k,q)}}}\right) c_k^{\downarrow} + \log\left(1 - e^{-\frac{\lambda_b}{m 2^{\min(k,q)}}}\right) c_k^{\uparrow} \\
&+ \sum_{k=1}^{q+1} \log\left(1 - e^{-\frac{\lambda_a + \lambda_x}{m 2^{\min(k,q)}}} - e^{-\frac{\lambda_b + \lambda_x}{m 2^{\min(k,q)}}} + e^{-\frac{\lambda_a + \lambda_b + \lambda_x}{m 2^{\min(k,q)}}}\right) c_{kk}
\end{aligned} \tag{60}$$

which is the two HyperLogLog case analog of (29). 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 (60) 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 (60) is separable into the sum of two log-likelihood functions that depend on λ_a and λ_b, respectively, and that follow (29). Hence, the joint maximum likelihood estimation of λ_a and λ_b gives the same results as estimating them independently by maximizing (29).

The first derivatives are

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \lambda_a}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{1}{m} \sum_{k=0}^{q} \frac{c_k^{\leftarrow} + c_{kk} + c_k^{\rightarrow}}{2^k} + \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a + \lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\rightarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\leftarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - 1}{e^{\frac{\lambda_a + \lambda_b + \lambda_x}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + 1} \frac{c_{kk}}{2^{\min(k,q)}}\end{aligned} \tag{61}$$

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \lambda_x}(\lambda_a, \lambda_b, \lambda_x | \vec{k}_1, \vec{k}_2) = &- \frac{1}{m} \sum_{k=0}^{q} \frac{c_k^{\rightarrow} + c_{kk} + c_k^{\downarrow}}{2^k} + \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_a + \lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\rightarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{1}{e^{\frac{\lambda_b + \lambda_x}{m 2^{\min(k,q)}}} - 1} \frac{c_k^{\downarrow}}{2^{\min(k,q)}} \\ &+ \frac{1}{m} \sum_{k=1}^{q+1} \frac{e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - 1}{e^{\frac{\lambda_a + \lambda_b + \lambda_x}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_b}{m 2^{\min(k,q)}}} - e^{\frac{\lambda_a}{m 2^{\min(k,q)}}} + 1} \frac{c_{kk}}{2^{\min(k,q)}}\end{aligned} \tag{62}$$

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

$$h_2 = \frac{x + 2h_1(1 - h_1)}{x + 2(1 - h_1)}. \tag{63}$$

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

$$\tilde{h}_2 = \frac{x + 2\tilde{h}_1(1 - \tilde{h}_1)}{x + 2(1 - \tilde{h}_1)} \tag{64}$$

The corresponding relative error ε_2 is given by

$$\varepsilon_2 = \frac{\tilde{h}_2}{h_2} - 1. \tag{65}$$

Putting (63) and (64) into (65) and using the first-order approximations

$$\frac{x + 2\tilde{h}_1(1 - \tilde{h}_1)}{x + 2h_1(1 - h_1)} = 1 + \varepsilon_1 \frac{2h_1\,(1 - 2h_1)}{x + 2h_1\,(1 - h_1)} + \mathcal{O}(\varepsilon_1^2) \tag{66}$$

and

$$\frac{x + 2(1 - h_1)}{x + 2(1 - \tilde{h}_1)} = 1 + \varepsilon_1 \frac{2h_1}{x + 2\,(1 - h_1)} + \mathcal{O}(\varepsilon_1^2), \tag{67}$$

we obtain

$$\varepsilon_2 = \varepsilon_1 \left(\frac{2h_1\,(1 - 2h_1)}{x + 2h_1\,(1 - h_1)} + \frac{2h_1}{x + 2\,(1 - h_1)} \right) + \mathcal{O}(\varepsilon_1^2) \tag{68}$$

By numerical means it is easy to show that

$$0 \leq \frac{2h(x)\,(1 - 2h(x))}{x + 2h(x)\,(1 - h(x))} + \frac{2h(x)}{x + 2\,(1 - h(x))} \leq 0.517 \tag{69}$$

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

$$|\varepsilon_2| \leq 0.517\ |\varepsilon_1| + \mathcal{O}(\varepsilon_1^2) \tag{70}$$

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

$$\hat{x} \sum_{k=0}^{q} \frac{c_k}{2^k} + \sum_{k=1}^{q} c_k h\left(\frac{\hat{x}}{2^k}\right) + c_{q+1} h\left(\frac{\hat{x}}{2^q}\right) = m - c_0 \tag{71}$$

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

$$\left|\tilde{h}(x) - h(x)\right| \leq \varepsilon_h h(x) \tag{72}$$

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

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{c_k}{2^k} + \sum_{k=1}^{q} c_k \tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right) + c_{q+1}\tilde{h}\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right) = m - c_0 \tag{73}$$

Due to (72) there exists some $\alpha \in [-\varepsilon_h, \varepsilon_h]$ for which

$$\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{c_k}{2^k} + (1+\alpha)\left(\sum_{k=1}^{q} c_k h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^k}\right) + c_{q+1}h\left(\frac{\hat{x}\,(1+\varepsilon_x)}{2^q}\right)\right) = m - c_0 \tag{74}$$

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

$$\begin{aligned}\hat{x}\,(1+\varepsilon_x)\sum_{k=0}^{q}\frac{c_k}{2^k} + (1+\alpha)\left(\sum_{k=1}^{q} c_k h\left(\frac{\hat{x}}{2^k}\right) + \frac{c_k}{2^k}\hat{x}\varepsilon_x\beta\right) + \\ + (1+\alpha)\left(c_{q+1}h\left(\frac{\hat{x}}{2^q}\right) + \frac{c_{q+1}}{2^q}\hat{x}\varepsilon_x\beta\right) = m - c_0\end{aligned} \tag{75}$$

Subtracting (71) multiplied by $(1+\alpha)$ from (75) and resolving ε_x gives

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

Using $|\alpha| \leq \varepsilon_h$, $\beta \geq 0$, and (41) the absolute value of the relative error can be bounded by

$$|\varepsilon_x| \leq |\varepsilon_h|\,\frac{(m - c_0) - \hat{x}\sum_{k=0}^{q}\frac{c_k}{2^k}}{\hat{x}\sum_{k=0}^{q}\frac{c_k}{2^k}} \tag{77}$$

Furthermore, using (38) we finally get

$$|\varepsilon_x| \leq |\varepsilon_h|\,\frac{\frac{1}{2}\sum_{k=1}^{q}\frac{c_k}{2^k} + \frac{c_{q+1}}{2^q}}{\sum_{k=0}^{q}\frac{c_k}{2^k}} \leq |\varepsilon_h|\left(\frac{1}{2} + \frac{\frac{c_{q+1}}{2^q}}{\sum_{k=1}^{q}\frac{c_k}{2^k}}\right) \leq |\varepsilon_h|\left(\frac{1}{2} + \frac{c_{q+1}}{m - c_{q+1}}\right) \tag{78}$$

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*, EDBT '08, pages 618–629, New York, NY, USA, 2008. ACM.

[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*, KDD '14, pages 442–451, New York, NY, USA, 2014. ACM.

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

[5] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM.

[6] Philippe Flajolet, ric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *IN AOFA 07: PROCEEDINGS OF THE 2007 INTERNATIONAL CONFERENCE ON ANALYSIS OF ALGORITHMS*, 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. ACM, 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), 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 Trans. Database Syst.*, 15(2):208–229, June 1990.

[13] Marianne Durand and Philippe Flajolet. *Algorithms - ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings*, chapter Loglog Counting of Large Cardinalities, pages 605–617. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

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