Cardinality Estimation of Distinct Items - An Overview

Sebastian Balsam

November 25, 2025

Abstract

In this paper I want to give an overview of different solutions for the Count Distinct Problem...

1 Introduction

The cardinality estimation problem or count-distinct problem is about finding the number of distinct elements in a data stream with repeated elements. These elements could be URL's, unique users, sensor data or IP addresses passing through a data stream. A simple solution would be to use a list and add an item to this list each time we encounter an unseen item. With this approach we can give an exact answer to the query, how many distinct elements have been seen. Unfortunately if millions of distinct elements are present in a data stream, with this approach, we will soon hit storage boundaries and the performance will deteriorate.

As we often do not need an exact answer, streaming algorithms have been developed, that give an approximation that is mostly good enough and bound to a fixed storage size. There different approaches to solve this problem. Sampling techniques are used as an estimate by sampling a subset of items in the set and use the subset as estimator, e.g. the CVM algorithm by Chakraborty et al. [Chakraborty et al., 2023]. In its simplest form if we take a sample size of N_0 for a set of size N, we get an estimate by counting the distinct items times N/N_0. While sampling methods generally give a good estimate, they can fail in certain situations. If rare elements are not in the sampled set, they would not be counted and we would underestimate. Another problem is replicating structures in the data. If we sample every 10th item, and accidentally every 10th item is different, while other items are mostly the same, we would overestimate. For database optimization, machine learning techniques have been used recently for cardinality estimation [Liu et al., 2015, Woltmann et al., 2019, Schwabe and Acosta, 2024].

The technique I want to focus on in this paper are stochastic sketch techniques that implement a certain data structure and an accompanying algorithm to store information efficient for cardinality estimation. I will follow the development chronologically from the first algorithm of Flajolet and Martin in 1985 [Flajolet and Martin, 1985], to the HyperLogLog algorithm in 2007.

TODO: Give an overview of the used techniques. Tell what I want to say and what I leave out.

2 Flajolet and Martin

Before the advent of the internet there were not many data stream applications as we have today. But databases existed and began to grow in size. As table sizes grew, evaluation strategies became important, how to handle such big data tables for join operations. Query optimizers were developed that could find the optimal strategy.

In this context Flajolet and Martin developed in 1985 a method to estimate the number of distinct items in a set in a space efficient way[Flajolet and Martin, 1985]. The idea behind the method is to use a hash function that maps n items uniformly to integers. If we look at the binary representation of these integers, and check the longest runs of zeros, in about $n/2$ cases, we have a '0' at any position. In about $n/4$ cases, we have two '0's consecutively. With a chance of $1/8$ we have three '0' in a row, and so on. That means if we start for example at the right and count the zeros for each element in the stream, based on maximal number we have seen so far, we can estimate the number of items. When we add an item, we use the algorithm as described in Algorithm 1 to add set the position of the rightmost '1' in a bitmap.

Flajolet and Martin found that they get better result, if an average of multiple (m) bins of bitmaps are used. This is equivalent with running an experiment multiple times to get results closer to the expected value. With a higher number of bins available for the average, the estimation quality raises, but at the same time more memory to store the bitmap is needed. They use the hash value to decide the bin (by using the modulo operation) and save the rightmost '1' of the binary representation in this bin.

Algorithm 1 Adding an item in the Flajolet-Martin algorithm.

$m \leftarrow$ Number of bins
function $\rho(val)$
  return position of the first 1 bit in val.
end function
function ADDITEM(x)
  $hashedx \leftarrow hash(x)$
  $\alpha \leftarrow hashedx \bmod m$
  $index \leftarrow \rho(hashedx \text{ div } m)$
  $BITMAP[\alpha][index] = 1$
end function

When an estimate is needed, an average of the leftmost zeros over all bins are used, as shown in algorithm 2. The magic number $\rho = 0.77351$ is used, as the expected Value of

Algorithm 2 Get an estimate in the Flajolet-Martin algorithm.

function QUERY
    $\rho \leftarrow 0.77351, S \leftarrow 0$
    max $\leftarrow 32$ ▷ 32 bit per bin
    for $i := 0$ to $m - 1$ **do**
        $j \leftarrow 0$
        while $BITMAP[i][j] == 1$ and $j < max$ **do**
            $j \leftarrow j + 1$
        end while
        $S \leftarrow S + j$
    end for
    return $int(m/\rho * 2^{S/nmap})$
end function

R - the position of leftmost zero in the bitmap is

$$\mathbf{E}(R) \approx \log_2 \rho n$$

We sum up the bins and as each bin only counted $1/m$ items, we estimate the number with

$$\frac{1}{\rho} m \cdot 2^{\frac{S}{m}}.$$

The results for this algorithm are shown in Figure 1. The memory consumption for the algorithm is $m * 32$. That means with $m = 256$, we can produce estimates with about 5% standard error and have to use $256 \cdot 32 = 8192$ bit $= 1$kB of memory to safely count up to 100 million items, before the quality decreases.

Figure 1 shows the results of my implementation of the FM algorithm of a cardinality up to one million items. We can see that the algorithm results in strong overestimation below 1500 items for $m = 265$. After that, we only have a deviation on average of about 1 percent of the true number of items. The solution for numbers below 1500 items is simply to count these items exactly in an array, and only use Flajolet & Martin above a certain number.

The value of m - the number of bins is important for accuracy. With $m = 265$, Flajolet and Marin report a bias of 0.0073 and a standard error of 4.65%. For $m = 16$, we have a bias of 1.0104, about the same, but a standard error of 19.63%. These numbers match my own experiments.

3 HyperLogLog, 2007

A first refinement of the Flajolet & Martin algorithm came in 2003 by Durand and Flajolet [Durand and Flajolet, 2003]. Later in 2007 it was further refined [Flajolet et al., 2007] to improve the accuracy even further to an algorithm called HyperLogLog.

When we try to estimate the number of items, in the FM85 algorithm, we look for the maximum number of '1' in the bitmap. All the other '1' before and any bits after this position are not used. So instead of saving the complete bitmap, it should be enough to just save the maximum number. In addition, since we have a maximum of 32 bits originally in the bitmap, we only need 5 bits ($2^5 = 32$) for this number to be stored for each bin. This is an additional memory reduction.

- splitting the hash function to fill multiple buckets with a subset of bits of the hash. - only use the smallest 70% of values (throw away high outliers) - harmonic mean

- Compressed FM and HLL [Scheuermann and Mauve, 2007]

4 HyperLogLog++, 2013

5 LogLogBeta, 2016

6 ExtendedHyperLogLog, 2023

https://arxiv.org/pdf/2106.06525

7 HLL-TailCut, 2023

https://topoer-seu.github.io/PersonalPage/csqjxiao_files/papers/INFOCOM

8 Conclusion

Table 1: Table showing the different methods...

Algorithm	Memory	Comment
FM85	23	clever
HLL	45	here
HLL++	23	23
ExHLL	35	fsd
HLL-TailCut	23	sdf

References

[Chakraborty et al., 2023] Chakraborty, Vinodchandran, and Meel (2023). Distinct elements in streams: An algorithm for the (text) book.

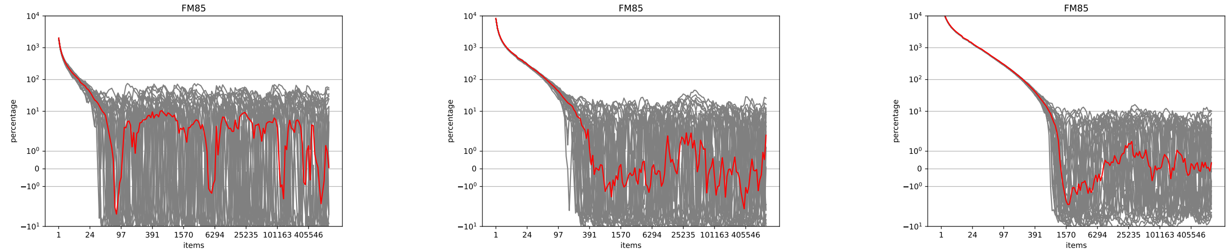
[Durand and Flajolet, 2003] Durand and Flajolet (2003). Loglog counting of large cardinalities.

[Flajolet et al., 2007] Flajolet, Fusy, Gandouet, and Meunier (2007). Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.

[Flajolet and Martin, 1985] Flajolet, P. and Martin, G. N. (1985). Probabilistic counting algorithms for data base applications.

[Liu et al., 2015] Liu, Xu, Yu, Covinelli, and ZuZarte (2015). Cardinality estimation using neural networks.

[Scheuermann and Mauve, 2007] Scheuermann and Mauve (2007). Near-optimal compression of probabilistic counting sketches for networking applications.



(a) m=16

(b) m=64

(c) m=265

Figure 1: The percentage of deviation from the true number of items over 30 runs (in grey) are shown (m=265) up to one million items. The red line shows the average over the 30 runs.

[Schwabe and Acosta, 2024] Schwabe and Acosta (2024). Cardinality estimation over knowledge graphs with embeddings and graph neural networks.

[Woltmann et al., 2019] Woltmann, Hartmann, Thiele, Habich, and Lehner (2019). Cardinality estimation with local deep learning models.