WROCŁAW UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF FUNDAMENTAL PROBLEMS OF TECHNOLOGY

FIELD: Big Data Analytics

MASTER OF SCIENCE THESIS

Set Operations on Count-Min Sketches

AUTHOR:
Krzysztof Agieńczuk

SUPERVISOR:
prof. dr hab. Jacek Cichoń

GRADE:

WROCŁAW 2020

Here's to the ones who dream
Foolish as they may seem

Contents

Chapter 1

Introduction

The goal of this thesis is to investigate the possibility of defining and conducting set operations on Count-Min Sketches.
Sketches are data structures which allow us to operate on a summary of seen data. This is predominantly needed when dealing with data streams, which recently becomes more and more common. Intuition behind what a data stream is can be that it is a continuous stream (flow) of incoming data, too large to store effectively. Naturally then, almost no function can be calculated on the whole stream as most of the times, memory needed to store such large amount of data is bigger than the memory available. For example Google processes 40000 searches every second (which accumulates to 3.5 billion per day) and there are 6000 tweets sent per second (500 million per day)[1]. This is clearly too much data to calculate even simplest aggregating functions exactly. That's why approximation is needed. Sketches are representative summary of a stream allowing for approximation some of the aggregating functions. Such need appears for example when monitoring Internet traffic or analysing and monitoring contents of big databases. Recently many different sketching algorithms have been proposed. Those include, but are not limited to Flajolet-Martin Sketch, Count Sketch, Count-Min Sketch or AMS Sketch. We will focus on the Count-Min Sketch (written also as CountMin, Cout Min or simply CMS).

Count Min Sketch is a probabilistic data structure which can represent a highly dimensional vector by hashing each element to a number of smaller vectors using simple hash functions. For more detailed explanation see Section 1.3 or [4]. The intuition is that increasing the codomain of the hash functions increase the accuracy of the estimate while increasing the number of functions decreases the probability of bad estimate.

The thesis consists of 4 chapters. This chapter is an introduction and short description of what a CMS are and how they can be used. In the second one, we shall present the theoretical analysis of chosen algorithms for merging, subtracting and multiplying sketches. Third one contains results of tests of said algorithms. Last chapter will be short summary and conclusions about work made and possible further development of those ideas.

1.1 Data Stream

Let's start with the data stream. It is a continuous flow of data that is recorded to be processed at the later stage. This can be either sensory data, timestamps, processed and prepared packets or something different. Often, the whole data in the stream far exceeds the memory available. This means that conventional data processing techniques, based on

persistence and persistent data structures, are not really useful. For that reason data has to either be processed "on the fly" or approximated. One trivial and well-known example of processing data on the fly is calculating the maximum. As calculating maximum does not require information about the whole dataset at any time (in contrast to say average), we can only store "biggest so far" and compare it against each incoming element. Size of the dataset influences only the time of calculations but not the amount of memory needed. We will not focus on this approach however, but rather on operating on already sketched data, where we don't have full dataset available.

Formally, data stream can be defined as a vector a, such that

$$a(t) = [a_0(t), ..., a_i(t), ..., a_n(t)]$$

with initial value of vector $\mathbf{a}$ being zero vector $\mathbf{0}$. Then each update is a pair (i_t, c_t) where i is an index and c_t is update value, meaning that

$$\begin{aligned} a_{i_t}(t) &= a_{i_t}(t-1) + c_t \\ a_{i'}(t) &= a_{i'}(t-1) \quad i' \neq i_t \end{aligned} \tag{1.1}$$

There are two possibilities when talking about update value. We can assume non-negativity of it or allow c_t to be negative. Those models are correspondingly called cash-register model and turnstile model.

1.2 Sketching

Sketching is an idea, that instead of trying to hold whole data stream in memory, we could somehow represent the data in shortened form. Such shortcut (sketch) obviously has to adhere to special requirements. Otherwise it would distort the data. Simplest scenario would be holding the average of incoming data. However, consider scenario when data consists of only 0's and 1's. Average would then be $\frac{1}{2}$ which doesn't even appear in the data and merely represents it. We can fix it by adding variance to the sketch, but again, we can think of an example for which this will not be enough. We also need to think about the practical "usefulness" of a sketch. It will not be useful, if eg. update procedure takes more time than we have between incoming elements.

If we list all our requirements we get the following list

- Reasonable amount of space used. Cormode suggests at most poly-logarithmic to n, where n is the size of the stream[5]
- Fast update and retrieve procedures
- Good accuracy guarantees
- As small dependence of accuracy guarantees on processing time as possible. Simply said it should be possible to shorten processing time without much accuracy losses.

Such data structures naturally don't allow for calculating the same algorithms one would use on the normal dataset. As such approximation is needed. There are some properties that can be calculated precisely, such as number of elements in the stream or maximal value, however most are not possible.

As mentioned, in this paper we will focus on Count-Min sketch, but some other worth to mention sketching algorithms include Flajolet-Martin skethc (FM), k-Minimum Value(KMV) or AMS sketch (first proposed in 1996 by Alon, Matias and Szegedy)[2].

1.3 Count-Min Sketch

Count-Min sketch is a probabilistic data structure represented by $d \times w$ matrix. Often, CM is notated as $CM(\epsilon, \delta)$ where ϵ is a bound for acceptable error and $1-\delta$ is a probability that result is within a given bound. In other words, estimated value $\hat{a}$ is in range $a \pm \epsilon$ with probability $1 - \delta$. Mentioned w and d parameters are calculated from ϵ and δ as follows

$$\begin{aligned} w &= \left\lceil \frac{e}{\epsilon} \right\rceil \\ d &= \left\lceil \ln \frac{1}{\delta} \right\rceil \end{aligned} \tag{1.2}$$

At the beginning, each entry in the array is set to zero. Then d pairwise-independent hash functions are chosen.

Remark For simplification of analysis and calculations, we will consider sketches with $d = 1$. This results in single line sketch, which can be then shown to be isomorphic to the KMV sketch. This is more deeply described in Section 2.1

1.3.1 Update Procedure

Update value is a pair (i_t, c_t) with i being index of an item to increase and c value by which it should be increased. In many cases c is set to be 1. Then for each $j \leq d$ hash function is calculated and according field in jth row increased by c. Formally this can be written as

$$count[j, h_j(i)] := count[j, h_j(i)] + c_t \tag{1.3}$$

where *count* is a $w \times d$ array and $1 \leq j \leq d$ iterates over d rows of said array.

1.3.2 Query Procedure

Query Procedure is done by calculating same hash functions as used in updating and taking minimum of the returned values. Mathematically this can be written as

$$\hat{a}_i = \min_j count[j, h_j(i)] \tag{1.4}$$

1.3.3 Analogy to Bloom Filter

It is also noteworthy to mention that when we discard counting in favour of just marking the existence of an element (in implementation this can be done by using array of bits instead of arrays of integers) and choose number of hashes $d = 1$ this becomes the well-known Bloom Filter. We also know that there exists natural union operation which simply takes bitwise OR function of the sketches and results in $S(A) \cup S(B) = S(A \cup B)$ with $S(A)$ and $S(B)$ being sketches of datasets A and B. Analogous "naive" operation can be defined for CMS, using the linear property of this sketch. This means simplest, naive union function can be defined as element-wise addition. Analogously naive cardinality estimation can be calculated as operation on sketches. This is described in more detail in Section 2.2

Chapter 2

Theoretical analysis

This chapter contains the description of used methodology and theoretical analytics of used algorithms.

2.1 Methodology overview

In all of our analysis we considered Count-Min Sketches with one-dimensional array of counts, i.e. $d = 1$ and one hash function. Because of that we also don't need to go in detail about hash function, as talking about function independence in this case is unnecessary. This results in single line sketch. This is similar to the k-Minimum Value sketch. This is a sketch, which contains only k smallest numbers and is described in-depth in [5].

Considering analysed sketch is $d = 1$, it is worth mentioning that if one would want to implement usable instance of this sketch, all algorithms would have to be multiplied. This means each proposed and tested algorithm has to be done independently for each row/each hash function. Apart from that it has to be also guaranteed that corresponding hash functions are the same. CMS with $d = 5$ rows would be just 5 sketches analysed here, defined with independent hash functions.

Our aim was to look into possibilities of estimating the cardinalities of big datasets and operations on said sets using only sketches. Mostly we were looking for cardinalities of operation on two streams, eg. *how many elements appear in both A and B?* The operations chosen to be investigated were

- intersection $A \cap B = \{x \in \Omega : x \in A \wedge x \in B\}$
- union $A \cup B = \{x \in \Omega : x \in A \vee x \in B\}$
- difference $A \setminus B = \{x \in \Omega : x \in A \wedge x \notin B\}$

Tests were conducted in statistically viable cohort. For given number of elements split between two data streams, sketches of size $1/512, 1/256, 1/128, 1/64, 1/32$ were used. For fewer number of elements (e.g. for 2^{15} or 2^{16}) also sizes $1/16, 1/8$ and $1/4$ were used.

2.2 Size Estimation

As already mentioned in Section 1.3.3, most distinct counting sketches have a natural union operation. This means there exists such an operation $\hat{\cup}$ that for datasets A and B

gives

$$S(A) \hat{\cup} S(B) = S(A \cup B) \tag{2.1}$$

where $S(A)$ denotes sketch of dataset A. Having that, one can easily calculate the intersection of two sketches using the inclusion-exclusion principle in form

$$|A \cap B| = |A| + |B| - |A \cup B| \tag{2.2}$$

This approach however degenerates quickly. To see that consider estimating intersection of three sets. This requires multiple operations, each of which carrying an error. Those errors will layer up and distort the estimate.

As we are dealing with one-dimensional sketch, we used order statistics. Then, the estimate of cardinality of the sketched dataset is given by

$$N\left[|A|\right] = \frac{k-1}{\tau} \tag{2.3}$$

where k stands for sketch size and τ stands for biggest value in the sketch (see e.g. [3]). This means that the expected value of the random variable $\frac{k-1}{\tau}$ is equal to the size of the stream A. In other words, we can say that the random variable $\frac{k-1}{\tau}$ is an unbiased estimator of the size of A. This comes from Order Statistics, described in more detail in Appendix B.

2.3 Union Estimation

Imagine we have two sketches, S_A and S_B, which we want to merge. As established, each sketch is holding k sorted hash values in interval $[0, 1]$ and we will denote the biggest, k-th value, as τ. The idea behind the union operation, suggested by Ting in [6], was to create a temporary set of hash values containing items from both sketches and use that set to estimate the size of union of the datastreams A and B.

First however we set a threshold

$$\tau_{min} = \min(\tau_A, \tau_B) \tag{2.4}$$

which will be the condition of "accepting" an item into a temporary sketch.

For convenience, let's introduce some notation. As already established we will denote largest value in a sketch as τ. We will also denote whole set of hash values (values of the sketch) as $h(S)$ with S being the sketch. A and B will denote datastreams.

We then create a set containing all elements from hash sets of sketches A and B, which are smaller than τ_{min}. If $h(S, x)$ denotes hash set which elements fulfil $\leq x$, then

$$h(S_A \hat{\cup} S_B) = h(S_A, \tau_{min}) \cup h(S_B, \tau_{min}) \tag{2.5}$$

With that the estimate of a union can be expressed as

$$N[|A \cup B|] = \frac{|h(S_A \hat{\cup} S_B)| - 1}{\tau_{min}} \tag{2.6}$$

This is because we treat our newly created set $h(S_A \hat{\cup} S_B)$ as a sketch of situation $A \cup B$ and apply Equation 2.3. Thanks to that, this result can be immediately extended to handle multiple unions.

2.4 Intersection Estimation

Similar idea stands behind the estimation of intersection. First we want to create a set which would be representation of intersected elements. In this case we can drop the $\leq \tau_{min}$ restriction when doing it, as we don't have to worry about the number of elements exceeding the size of the sketch. As such, set of hash values representing the intersection can be simply defined as

$$h(S_A \hat{\cap} S_B) = h(S_A) \cap h(S_B) \tag{2.7}$$

This however allows for a situation when the threshold, denoted as τ_{min} is not in a new set. This is taken into account by defining helper function $\delta(S_A, S_B)$

$$\delta(S_A, S_B) = \begin{cases} 1 \; if \; \tau_{min} \in h(S_{A \hat{\cap} B}) \\ 0 \; if \; \tau_{min} \notin h(S_{A \hat{\cap} B}) \end{cases} \tag{2.8}$$

Now the estimate can be calculated. Situation is analogous to Equation 2.6, with the exception that we cannot be sure that our τ_{min} will be included in the newly created set. If it will, δ function will return 1, as in Equation 2.3. If will not, then we don't want to subtract anything, because we divide by the "next" number. We could say we use next order statistic.

$$N[|A \cap B|] = \frac{|h(S_A \hat{\cap} S_B)| - \delta(S_A, S_B)}{\tau_{min}} \tag{2.9}$$

2.5 Difference Estimation

Estimating the difference was suggested and tested in two different approaches. First idea was to estimate the difference using set operations and estimation of intersection. The second idea was to create the sketch which would be then used to estimate the size of a stream.

2.5.1 Difference via intersection

As mentioned above, one of two ways of estimating the difference was using the set operation and intersection operation. The whole idea can be expressed as follows.

$$N[|A \setminus B|] = N[|A|] - N[|A \cap B|] \tag{2.10}$$

This results in error rate comparable to error in estimation of intersection as shown on Figures 3.1, 3.2 and 3.3.

2.5.2 Difference via sketch creation

Second approach to estimate the difference is analogous to the approaches shown in Sections 2.3 and 2.4. We start by creating a sketch of hash values which we will treat as a "normal" sketch and estimate the size of dataset using method shown in Section 2.2.

The sketch is created as

$$h(S_{A \setminus B}) = h(S_A) \setminus h(S_B) \tag{2.11}$$

Then we can treat it as normal sketch and use Equation 2.3 to estimate the size. However, we have no guarantee that τ_{min} will end up in the resulting sketch. That's why we decided to define additional function $\delta(S_A, S_B)$

$$\delta(S_A, S_B) = \begin{cases} 1 \; if \; \tau_{min} \in h(S_{A \setminus B}) \\ 0 \; if \; \tau_{min} \notin h(S_{A \setminus B}) \end{cases} \tag{2.12}$$

This results in formula

$$N[|A \setminus B|] = \frac{|h(S_A) \setminus h(S_B)| - \delta(S_A, S_B)}{\tau_{min}} \tag{2.13}$$

We decided to test both approaches, using the δ function, as well as using 1 in all cases. As shown later, in Chapter 3, results were comparable between all three approaches and the differences between them were insignificant.

Chapter 3

Results

In this chapter we will present the results of our work and conducted tests. All tests were conducted in a statistically viable cohort and the results were averaged. Used sketch sizes were $1/512, 1/256, \ldots$ etc. with different amount of sketches for different sizes (due to time of calculation) but always following the same pattern. Amount of tested sketches is visible on the figures. We calculated mean error for each sketch size and variance of said error. Used datastream sizes were 2^{15}, 2^{16} and 2^{19} multiplied by 1.5. This 1.5 factor was skipped in the plot titles.

The implementation and tests were done in Scala programming language.

3.1 Accuracy of operations

Three chosen operations were tested in accordance with the description above. As one can see on the Figure 3.1 the accuracy grows with the size of sketch.

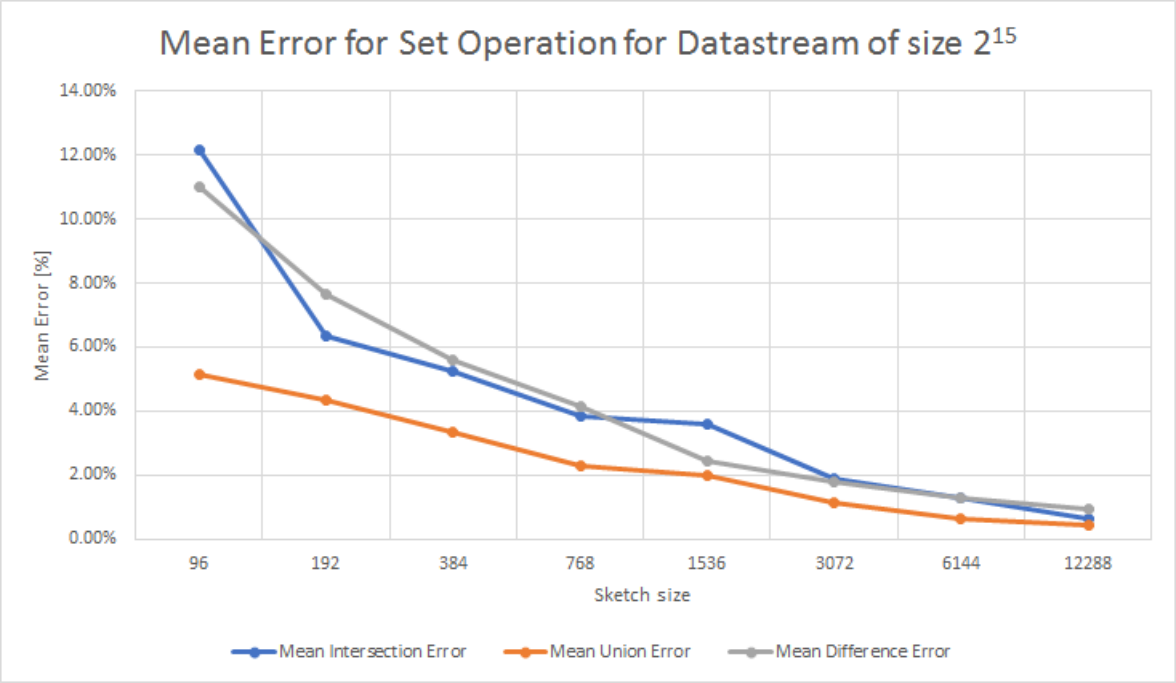

Figure 3.1 Mean error for different Set Operations for Datastream of size 2^{15}

This trivial observation is followed by two other, more significant. Firstly - increasing the size of a sketch leads to better estimations, but only up to a point, after which

increase in accuracy slows down rapidly and becomes practically unprofitable. In Figures 3.1 and 3.2 we can see rapid increase in first few sketches (12% to 4% error of estimation in Figure 3.1 for intersection), then a slow down and no significant difference between biggest sketches. In Figure 3.3 there is no slow down at the end, because biggest sketches were omitted due to time of calculation and previous experience with lack of significant improvement. Only sketches of sizes 1/512,...,1/32 were used (≈0.2% up to ≈3% of original datastream size).

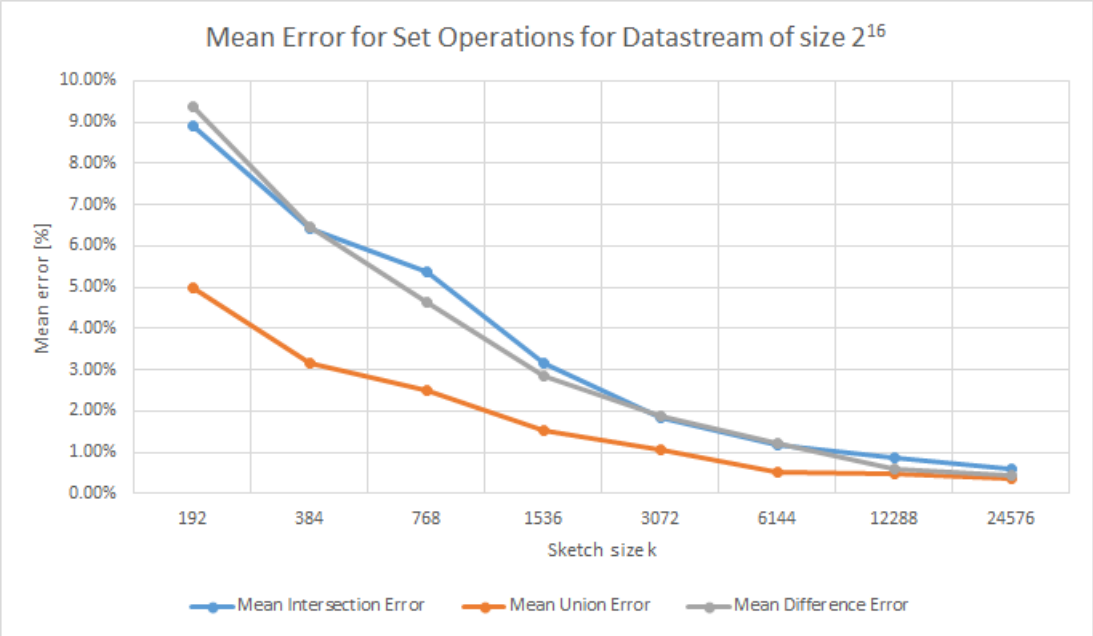

Figure 3.2 Mean error for different Set Operations for Datastream of size 2^{16}

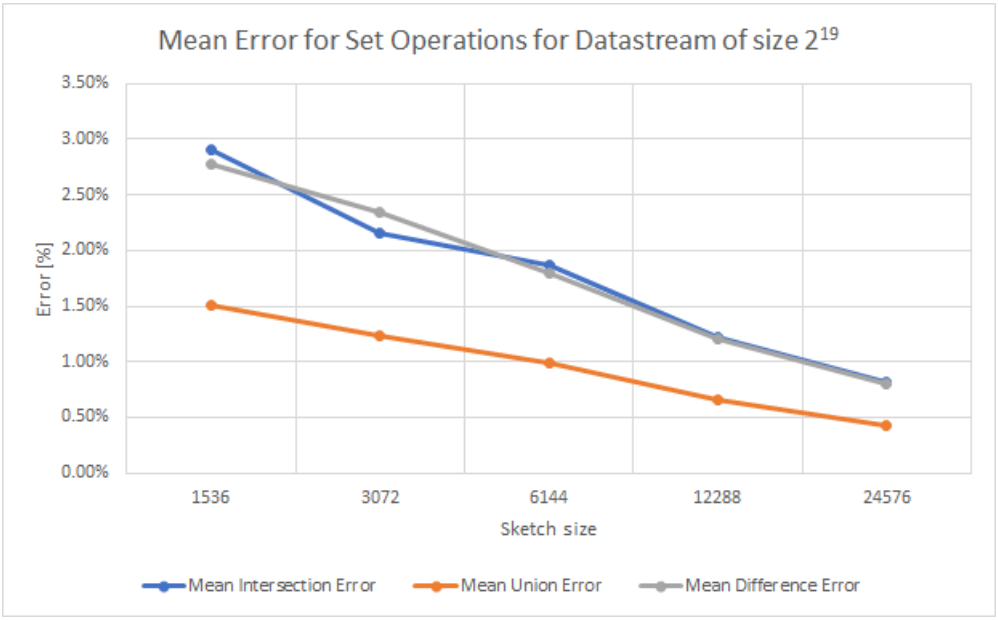

Figure 3.3 Mean error for different Set Operations for Datastream of size 2^{19}

Second interesting observation can be drawn from comparing results from different test cases. The same relative size of sketch (say, 1/512 of the size of datastream) yields better

results for bigger datastreams. This is increase in estimation accuracy is caused also by bigger absolute size of a sketch, but leads to a conclusion, that for bigger datastreams, smaller percent of elements needs to be stored to achieve expected accuracy. This effect has been shown on Figure 3.4. Keep in mind, that relative sizes (sketch size/datastream size) are compared. Absolute sizes vary.

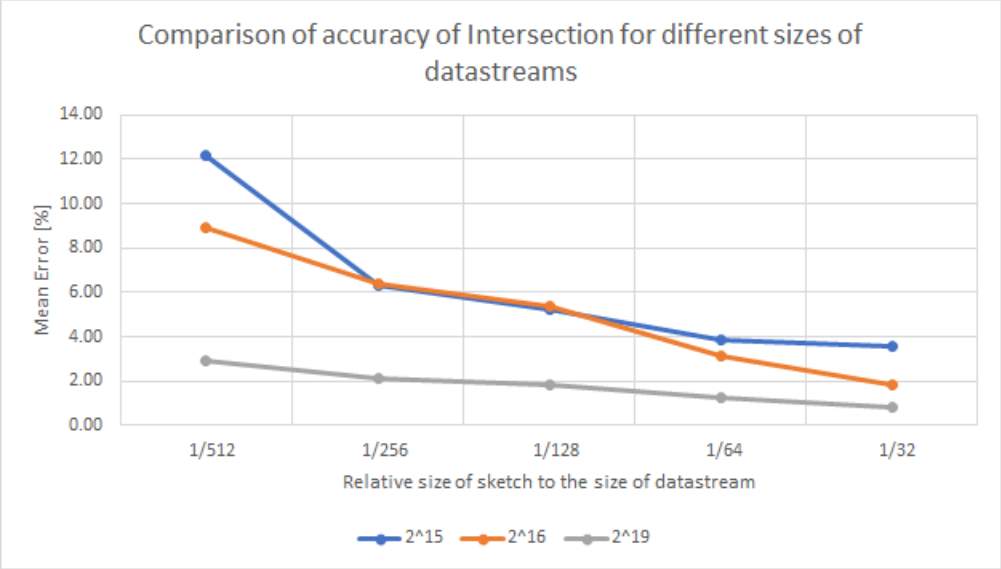

Figure 3.4 Comparison of accuracy of Intersection for different sizes of datastreams

The same situation can be seen also for other operations.

Also variance of the averaged error was calculated. From Figures 3.5 and 3.6 we can see rapid drop off in variance. This means that even small sketches at around 1% of datastream size (on Figure 3.5 this is sketch size = 384) allow to achieve reasonable accuracy with good variance.

Situation looks a little bit different on Figure 3.7. Similarly to error in estimation, we see smaller starting value and slower drop. This supports our earlier conclusion that for bigger datastream we can achieve good estimation with smaller relative size of a sketch.

3.1.1 Different approaches to the difference

As mentioned in Section 2.5, three different approaches have been suggested and tested. The tests were made on datastreams of 2^{15} elements, of which 1/3 were common, i.e. existing in both streams. On Figure 3.8 we can see a comparison of those three algorithms. It is clearly visible that the differences between them are not significant and decrease with the size of the sketch.

As the results presented are very close to each other, we decided to present them, in averaged form, additionally in a table below.

3.2 Accuracy for different relative sizes

As all other tests were conducted on datastreams that had $\approx \frac{1}{3}$ of common elements, we decided also to test the influence of relative size of intersection on the accuracy. The results were shown on Figure 3.9. This test was conducted for datastreams of size 2^{15}.

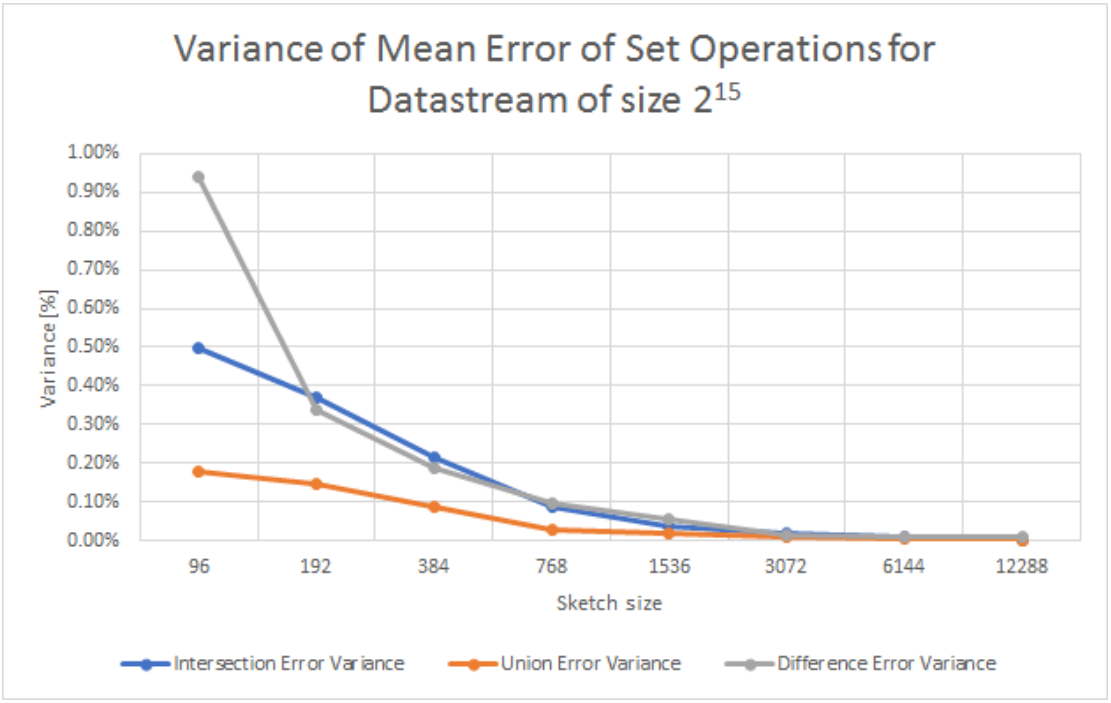

Figure 3.5 Variance of Mean Error of Set Operations for Datastream of size 2^{15}

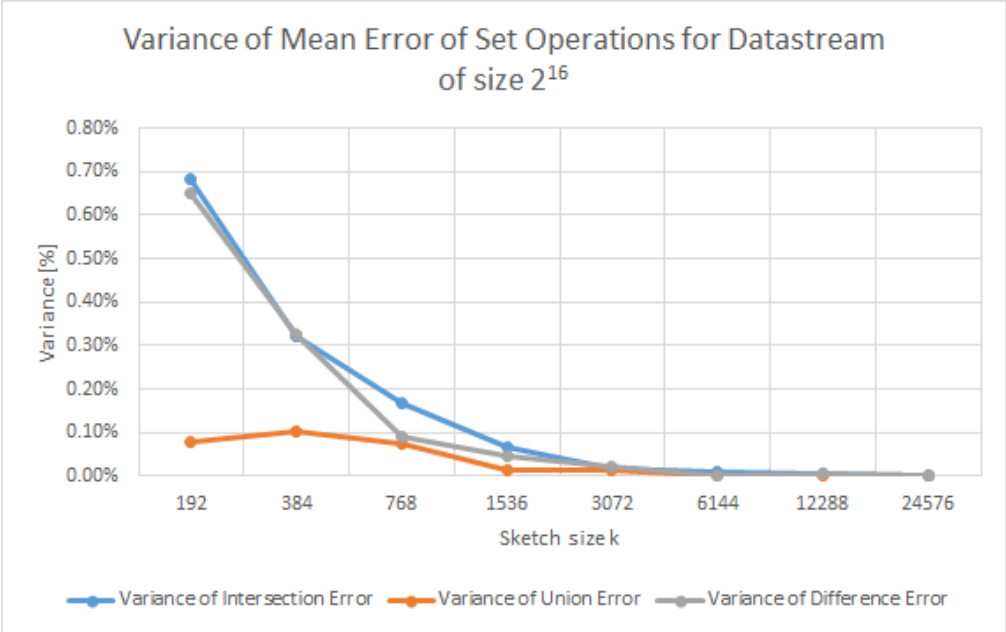

Figure 3.6 Variance of Mean Error of Set Operations for Datastream of size 2^{16}

Table 3.1 Results of estimating the difference by 3 algorithms compared

k	Via intersection	Using delta function	Analogous to standard estimator
128	11.17%	11.43%	10.89%
256	9.11%	9.25%	8.82%
512	5.30%	5.36%	5.16%
1024	3.55%	3.57%	3.49%
2048	2.31%	2.29%	2.30%

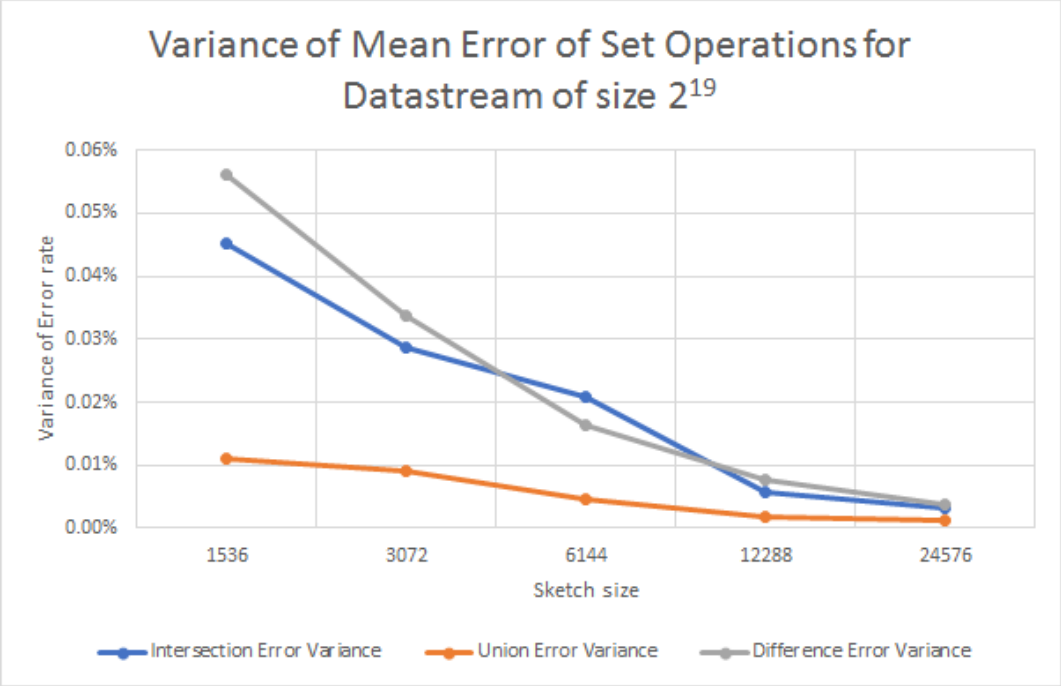

Figure 3.7 Variance of Mean Error of Set Operations for Datastream of size 2^{19}

The probability means the probability that during creation of datastreams, generated element will be assigned to both datastreams instead of one.

$$Probability = \Pr(x \in A \wedge x \in B) \tag{3.1}$$

So, the bigger the probability, the bigger the intersection of both datastreams. The smallest intersection size is shown on the right.

From Figure 3.9 one can see that, the smaller the common part gets, the less accurate the estimation is. This effect can be minimised by using bigger sketches. $k = 1024$ which is $\frac{1}{32}$ of a datastream size looks much more robust than $k = 128$ ($\frac{1}{256}$ of datastream size). However already $k = 512$ appears to give rather reliable estimation, with error not escaping above 10%.

3.3 Increase in accuracy

Those results pose a question about the ways of increasing the accuracy of estimations. Obviously the most natural is to increase the size of a sketch. This however cannot be done infinitely. Not only because of memory constraints, but also because of time complexity. Consider a continuous datastream. Every tick (defined period of time) new data comes in. This new item has to be hashed, prepended to the sketch and then "swapped" to the right place to keep the sketch in sorted order to be sure that last element τ_{min} is a threshold[1]. This results in $O(k)$ time complexity for sketch of size k. This means that we have to be able to conclude the update procedure in time smaller than the average time between new data coming in. This puts a natural barrier on the size of a sketch. In a

[1]This idea of "swapping" is the most efficient solution in this case, as in worst case we would need to do n "swaps" for sketch of size n. This is reminiscent of bubble sort algorithm, but we keep complexity to $O(n)$ as we are sorting only one element. Alternatively one can use another sorting algorithm, but this increases the complexity to at least $O(n \log n)$

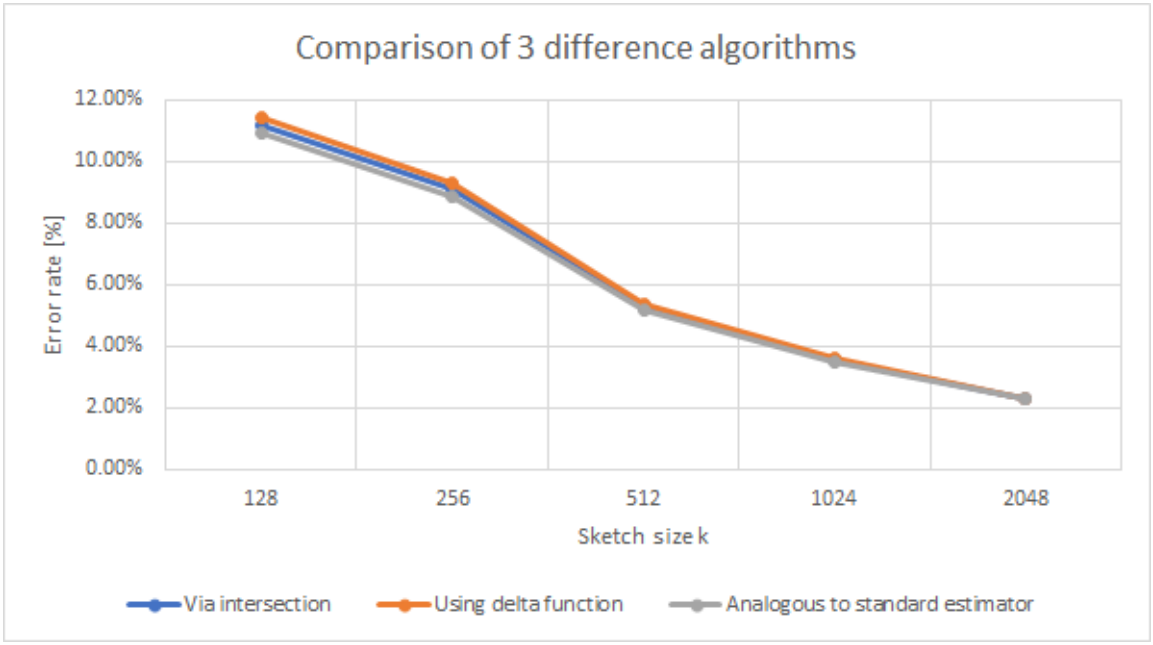

Figure 3.8 Comparison of different methods for Difference Operation

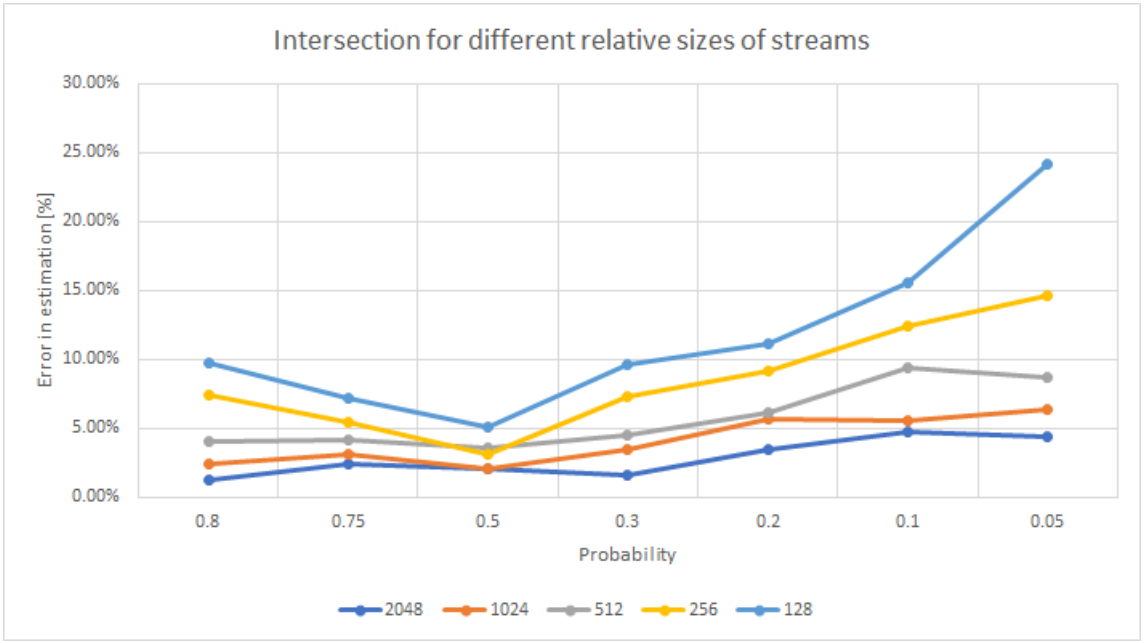

Figure 3.9 Accuracy of intersection operation for different relative sizes of datastreams

case with more hash functions this barrier gets even lower. One solution for this case can be simultaneous operation on each sketch. This allows to keep the size, while having a benefit of more sketches which provides higher accuracy as shown in Cormode[5].

As already mentioned above, another option to increase the accuracy of an estimation, we can also increase the size of a sketch. This is visible on Figures 3.1, 3.2 and 3.3 in Section 3.1. This is of course a trivial solution to a complex problem, which causes problems as described above, but nevertheless it is important to notice that small increase in sketch size can give a significant improvement. Also, increase in dataset size allows for smaller sketch. From Figure 3.9 we see that also different proportion of common elements in the streams influence the accuracy of estimation. Playing with this three factors one can aim to achieve better performance.

Chapter 4

Summary and conclusion

Within the scope of this thesis possibilities of conducting set operations on Count Min Sketches were investigated. We implemented algorithms for intersection and union suggested by Ting[6] to check their accuracies. They proved to be a good estimates for bigger datasets with accuracy increasing with the size of the dataset. In almost all of our test cases we achieved mean error rate below 10%. We also suggested a few ways to calculate the estimation difference of datastreams and found no significant differences between them (Figure 3.8) as well as that their accuracy is of same order as those mentioned earlier (Figures 3.1, 3.2 and 3.3). We have seen rapid decrease in mean error rate with small increases in sketch sizes in the lower end of tested sizes. We also noticed lower mean error rates for the same relative sketch/datastream size when datastream was bigger. This leads to conclusion that for bigger datastreams, smaller part of the data has to be kept in order to estimate the size of it. In Section 3.3 we also looked into other ways to increase the accuracy of the sketch and into the bounds on the size of a sketch.

Appendices

Appendix A

Chosen code listings

In this chapter, the code used for conducting the simulations is presented. Those listings come from the `ImprovedSingleLineCMS` class.

On the Listing A.1 the code for adding an element to a sketch has been presented. The `sketch` in terms of a program was defined as an `ArrayBuffer`.

Listing A.1 Method for adding an element to a sketch

```
def addElement(element: Float): this.type = {
   if (!sketch.contains(element) && element < sketch(k - 1)) {
      sketch.prepend(element)
      sketch.remove(k)
      sketch = swapSort(sketch)
   }
   this
}
```

Method `swapSort` is the algorithm mentioned in Section 3.3. It's actually bubble sort for a single element, but in this case, as we have an array that's sorted up to one element, this gives us the best running time, of complexity $O(n)$.

Listing A.2 Swapping sort

```
def swapSort(sketch: ArrayBuffer[Float]): ArrayBuffer[Float] = {
   var temp: Float = 0f
   for (i <- 0 until sketch.size - 1) {
      if (sketch(i) > sketch(i+1)) {
         temp = sketch(i+1)
         sketch(i+1) = sketch(i)
         sketch(i) = temp
      }
      else sketch
   }
   sketch
}
```

Below, on Listing A.3 code for estimating the intersection of two datastreams has been presented. This is code for the Equation 2.9. Analogically other operations were converted to Scala code but were skipped here for brevity. Whole source code used for the tests is available on the attached CD. `SingleLineCMS` is a Scala trait, which is implemented by the `ImprovedSingleLineCMS` class, hence `override` modifiers.

Listing A.3 Method for calculating the intersection estimation

```
override def intersection(secondSketch: SingleLineCMS, tauMin: Float):
    Set[Float] = sketch.toSet.intersect(secondSketch.getSketch.toSet)

override def intersectionEstimation(secondSketch: SingleLineCMS, tauMin:
    Float): Float = {
   val intersectedSet: Set[Float] = intersection(secondSketch, tauMin)
   if (intersectedSet.contains(tauMin)) (intersectedSet.size - 1)/tauMin
   else intersectedSet.size/tauMin
}
```

We present also the code for estimation of difference using the "via intersection" method (Listing A.4), because this was most commonly used in our tests. Other methods are available in the full version of the code on an accompanying CD. All estimation methods were placed in a class, and take other instance of a `SingleLineCMS` and `Float tauMin` as their parameters.

Listing A.4 Method for calculating the difference estimation

```
override def differenceEstimation(secondSketch: SingleLineCMS, tauMin: Float):
    Float = Math.abs((sketch.size - 1)/tauMin -
    intersectionEstimation(secondSketch, tauMin))
```

Appendix B

Order statistics

Order statistic is basically sorted (ordered) in increasing order set of random variables. Having a sample $X_1, X_2, ..., X_n$, we denote order statistic as $X_{(1)}, X_{(2)}, ..., X_{(n)}$. Note that order label has nothing to do with random variable label, i.e. X_1 doesn't necessarily equal $X_{(1)}$. Because we sort random variables, $X_{(1)} = \min\{X_1, X_2, ..., X_n\}$ and $X_{(n)} = \max\{X_1, X_2, ..., X_n\}$.

In our case we are dealing with random variable from uniform distribution on interval $[0, 1]$. From probability theory we know, that probability density function for k-th Order Statistic is

$$f_{(k)}(x) = k\binom{n}{k}x^{k-1}(1-x)^{n-k} \tag{B.1}$$

Expected value is defined as $\int xf(x)dx$. Using PDF from Equation B.1 we can now calculate expected value of k-th Order Statistic, which we will denote by $U_{(k)}$.

$$E\left[U_{(k)}\right] = \int_0^1 (x)\left[k\binom{n}{k}x^{k-1}(1-x)^{n-k}\right]dx = k\binom{n}{k}\int_0^1 x^k(1-x)^{n-k}dx \tag{B.2}$$

We see, that function under the integral follows Beta distribution with parameters $\mathbf{B}(k+1, n-k+1)$. As such, we know the solution to this integral

$$E\left[U_{(k)}\right] = k\binom{n}{k}\int_0^1 x^k(1-x)^{n-k}dx = k\binom{n}{k}\frac{k!(n-k)!}{(n+1)!} = \frac{k}{n+1} \tag{B.3}$$

To estimate N (random variable, being the size of the stream) however, we will need to calculate the estimated value of inverse of $U_{(k)}$

$$\begin{aligned} E\left[\frac{1}{U_{(k)}}\right] = \int_0^1 \frac{1}{x}\left[k\binom{n}{k}x^{k-1}(1-x)^{n-k}\right]dx &= k\binom{n}{k}\int_0^1 x^{k-2}(1-x)^{n-k}dx \\ &= k\binom{n}{k}\frac{(k-2)!(n-k)!}{(n-1)!} = \frac{n}{k-1} \end{aligned} \tag{B.4}$$

As we want to estimate N, after simple manipulation, we get $N = (k-1)\frac{1}{U_{(k)}}$. Therefore estimated value of the size of N will be

$$E[N] = (k-1)E\left[\frac{1}{U_{(k)}}\right] = (k-1)\frac{n}{k-1} = n \tag{B.5}$$

Variance can be calculated from standard formula $Var(X) = E[X^2] - E[X]^2$. For that we will need to calculate the second moment.

$$E[N^2] = (k-1)^2 E\left[(\frac{1}{U_{(k)}})^2\right] \tag{B.6}$$

To calculate Equation B.6 we will need to evaluate $E\left[(\frac{1}{U_{(k)}})^2\right]$.

$$E\left[(\frac{1}{U_{(k)}})^2\right] = \int_0^1 \frac{1}{x^2}\left[k\binom{n}{k}x^{k-1}(1-x)^{n-k}\right]dx = k\binom{n}{k}\int_0^1 x^{k-3}(1-x)^{n-k}dx$$
$$= k\frac{n!}{k!(n-k)!}\mathbf{B}(k-2, n-k+1) = k\frac{n!}{k!(n-k)!}\frac{(k-3)!(n-k)!}{(n-2)!} = \frac{n(n-1)}{(k-1)(k-2)} \tag{B.7}$$

After doing that we can substitute Equation B.7 back into Equation B.6 to get

$$E[N^2] = (k-1)^2\frac{n(n-1)}{(k-1)(k-2)} = \frac{n(n-1)(k-1)}{k-2} \tag{B.8}$$

Now, variance is

$$Var(N) = \frac{n(n-1)(k-1)}{k-2} - n^2 = \frac{n^2 - n(k-1)}{k-2} \tag{B.9}$$

We can also calculate correlation coefficient, which is defined as a standard deviation divided by expected value.

$$\begin{aligned} CC(N) = \frac{\sqrt{\frac{n^2-n(k-1)}{k-2}}}{n} &= \sqrt{\frac{n^2-n(k-1)}{k-2}\frac{1}{n^2}} = \sqrt{\frac{n^2-n(k-1)}{n^2(k-2)}} \\ &= \sqrt{\frac{1}{k-2} - \frac{k-1}{n(k-2)}} < \sqrt{\frac{1}{k-2}} \end{aligned} \tag{B.10}$$

The last inequality holds, as both k and n are positive. This also means, that the accuracy is independent of the size of datastream.

Bibliography

[1] Internet live stats. https://www.internetlivestats.com/, Mar. 2020.

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

[3] K. Beyer, P. Haas, B. Reinwald, Y. Sismanis, R. Gemulla. On synopses for distinct-value estimation under multiset operations. strony 199–210, 01 2007.

[4] G. Cormode. Sketch techniques for approximate query processing. *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases. NOW publishers*, 2011.

[5] G. Cormode, S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58 – 75, 2005.

[6] D. Ting. Towards optimal cardinality estimation of unions and intersections with sketches. strony 1195–1204, 08 2016.

List of Figures