Approximate Counting Using a Fixed Probability Counter And Csurös Counter

Pedro Marques, 92926

Resumo **– Este relatório apresenta dois metodos de contadores, um com probabilidade fixa e um floating point counter proposto por Miklós Csuro, desenvolvidos utilizando Python3. O objetivo é contar a ocorrência de cada letra em diferentes ficheiros de texto de linguas diferentes. Assim, tenciona-se verificar um correto armazenamento de uma determinada contagems, utilizando menos espaço computacional, nomeadamente diminuindo o número de bits necessário para armazenar estas contagens.**

Abstract **– This report presents two counter methods, one with fixed probability and another using a floating-point counter proposed by Miklós Csurös, developed using Python3. The main goal is to count the frequency of each different letter in varied text files of different languages. It is expected that these algorithms will use lesser memory, by reducing the amount of bits necessary to count a number of events.**

I. INTRODUCTION

In certain computational tasks, the usage of counters is essential for the overall process. As such, many of these tasks utilize more than one counter and therefore end up using a lot of memory space.

II. APPROXIMATE COUNTING

Approximate counting has the objective of reducing the memory footprint of a large number of events. It works by only increasing the counter when a certain probability is met. As such, it cannot provide exact results for the counter's exact real value but only an estimate. Morris introduced the Approximate Counting Algorithm in 1977 with his paper *Counting Large Numbers of Events in Small Registers.*

A. Morris' Solution

With the intention of keeping the program fast and the used memory low, Morris started by proposing a simple algorithm using a 'flip a coin' method.

As such, for each event, a coin would be flipped, only if it was *heads* would the counter increase. This simple solution allowed for a low margin of error where the expected final value of the counter would be around ½ of the real number of events. This method did not prove useful for small counts. In this case, the error was really high not proving itself useful on all accounts.

This is what is called a fixed probability counter, in this case, with a 50% probability.

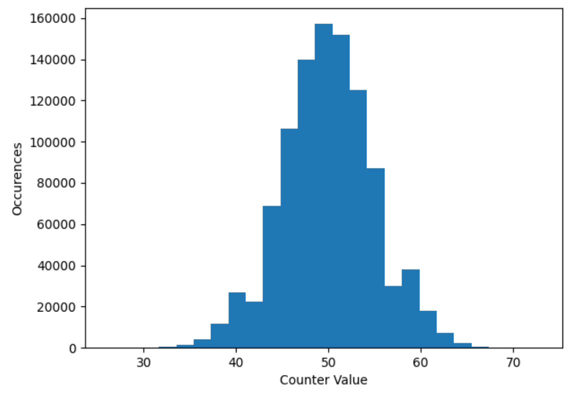


Figure 1 – Distribution of Counter Values Using The Coin Flip Method

In the figure above, we can visualize the usual probability of the counter values for the previously mentioned examples. In this case, 100 real events are registered while using an exact counter. Using the fixed probability of ½, the estimated counter value is 50. As we can see, over 1,000,000 trials were conducted and it is obvious that the registered values tend to this estimate.

By abandoning the idea of simply flipping a coin, Morris generalized this simple counter for any base ½ probability. As such, a simple counter initialized as X=0 is incremented by one with a generalized probability of 2^{-X}.

Another generalization of Morris' algorithm includes the *q-ary counter.* Two variables are necessary for this method: some $r \geq 1$ and $q = 2^{1/r}$. Using the last one, the probability of incrementing the counter is simply q^{-X}, or simply $2^{-X/r}$.

$$f(x) = \frac{q^x - 1}{q - 1} = \frac{2^{x/r} - 1}{2^{1/r} - 1}.$$

Figure 2 - Actual Event Count for a q-ary Counter

B. Csurös Algorithm

Based on this study, Miklós Csurös proposed a novel counter algorithm for approximate counting. For a certain nonnegative integer d, M=2^d represented the tradeoff between memory usage and accuracy. The higher the d, the lower was memory usage and accuracy of the counter.

```
FP-Increment($X$)                          // returns new value of $X$
1 set $t \leftarrow \lfloor X/M \rfloor$   // bitwise right shift by $d$ positions
2 while $t > 0$ do
3     if RandomBit() $= 1$ then return $X$
4     set $t \leftarrow t - 1$
5 return $X + 1$
```

Figure 3 – Csurös Incrementer Pseudo-Code

Above is represented the pseudo-code for the counters incrementing algorithm. The variable t is the floor value of the division between the counter's current state and the previously determined M value. The loop initialized in the second line is not executed until *t>0*. As such, we can conclude that the algorithm is deterministic for the first *M-1* steps and will always increment the counter value during these, providing good accuracy for small counts.

The counter X's value using Csurös method can be estimated using $X = 2^d * t + u$, where u denotes the lower d bits. Furthermore, the actual count f(X) can be estimated using $f(X) = (M+u)*2^t - M$.

II. RESULTS

Three algorithms were implemented in Python3 to test the accuracy and memory efficiency of these counters. First of all, a simple exact counter was created, this counted exactly how many letters appeared in each tested text file. Furthermore, a fixed probability counter was developed using a 1/32 probability and a Csurö counter was also implemented and tested using d=4 and d=8.

Four text files were used during testing, these contained the literary work of Charles Dickens: Oliver Twist in different languages: English (En), French(Fr), German(Ge), and Dutch(Ne).

A. Exact Counter

The exact counter is really simple, a single dictionary contains all letters and the associated values represent the occurrences of each letter.

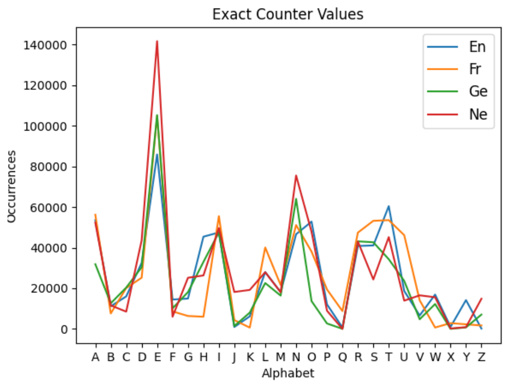
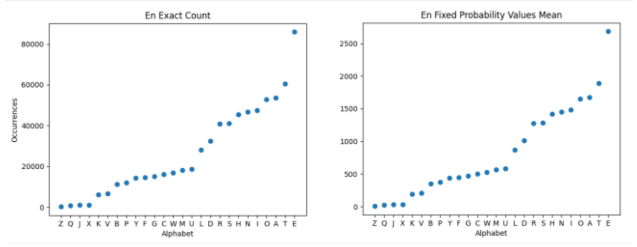


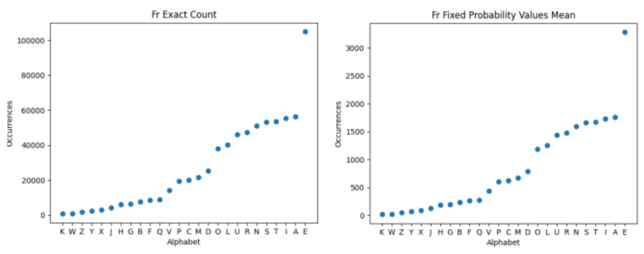
Figure 4 - Letter Occurrences Visualized

As presented, a similar letter distribution is present over the four text files. It is also obvious that some letters have a higher frequency than others. It is expected that the frequency distribution will remain similar in the next counters so that the letters with the highest frequency in the exact counter will be mostly the same in the fixed probability counter and in the Csurös Counter algorithms.

B. Fixed Probability

The fixed probability counter was based on a probability of 1/32. As such, the counter would only increment if such probability was met. Given this, the expected values were easy to calculate, given the exact counter values. Below are represented the comparison between the previously determined exact counts and the counter values with fixed probability 1/32.





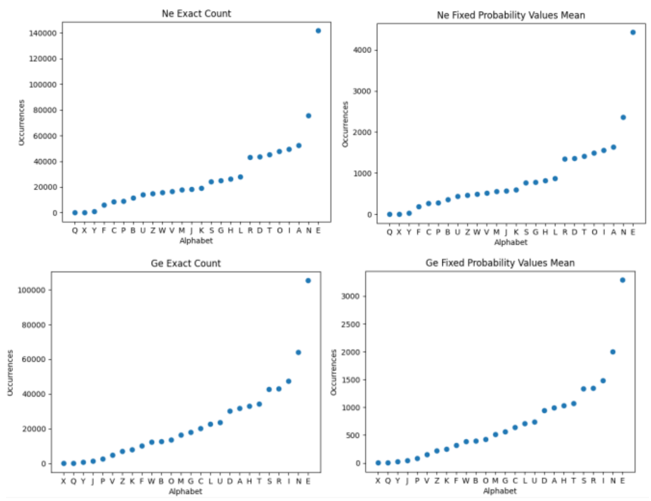
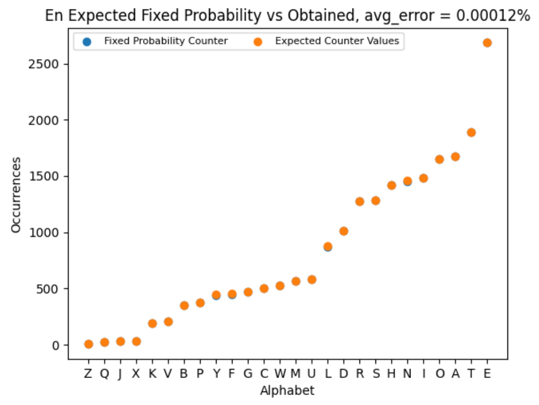
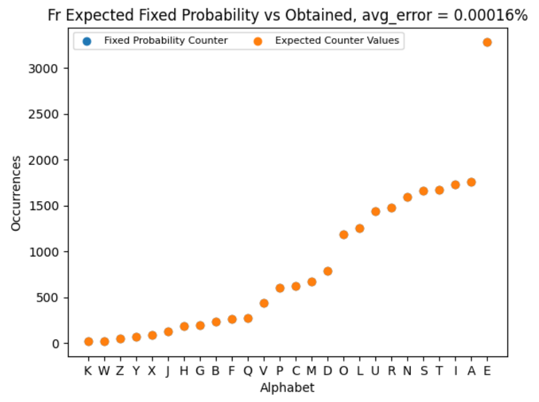
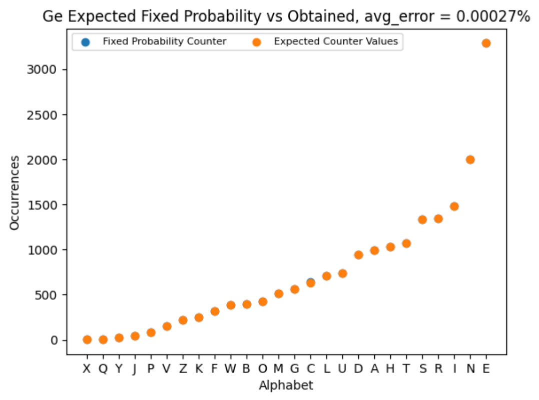


Figure 5 - Exact Counter Values Vs Fixed Probability Counter Values

As we can see, the counters retain the same letter frequency in the text files is represented in the same way using an exact counter or fixed probability counter. Furthermore, the expected results using this counter were compared to the obtained results







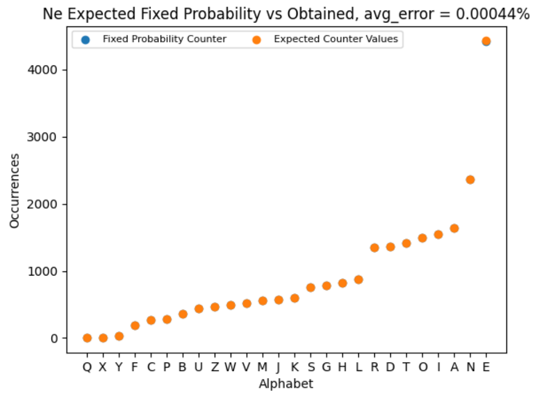


Figure 6 – Expected Counter Values vs Obtained Counter Values using Fixed Probability Counter (p = 1/32)

The fixed probability counter showed it was a great option in reducing memory usage while maintaining accuracy, the highest relative error being of only 0.00044% using the Dutch text file.

C. Csurös Counter

In this implementation of the Csurös Counter, the d factor was initially defined as 4. Seen that d is supposed to be the tradeoff between accuracy and memory efficiency, tests were conducted to analyze the error. For d=4, the error was unbearable, as expected, for n<2^d events, the count was precise, but considering larger events, the error grew.

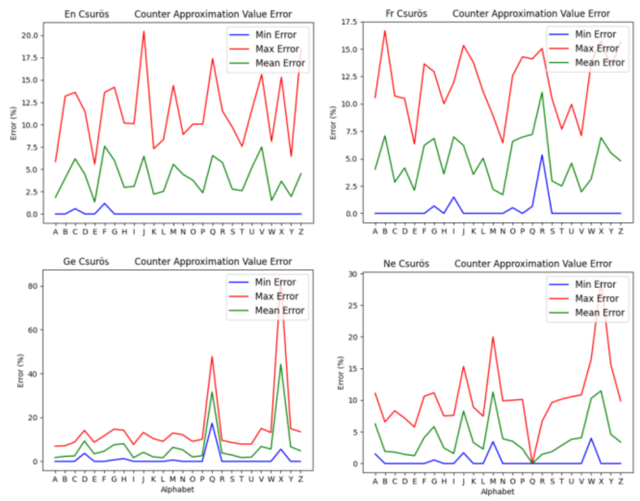


Figure 7 - Csurös Counter Value Error, d=4

For the different languages, with d = 4, the error between the estimated counter value and obtained is great. In some cases, it can reach a maximum of 85% but for the most part, it is between 0-10%. Even though this last interval might not seem much, the error further increases when we try to predict the real exact value.

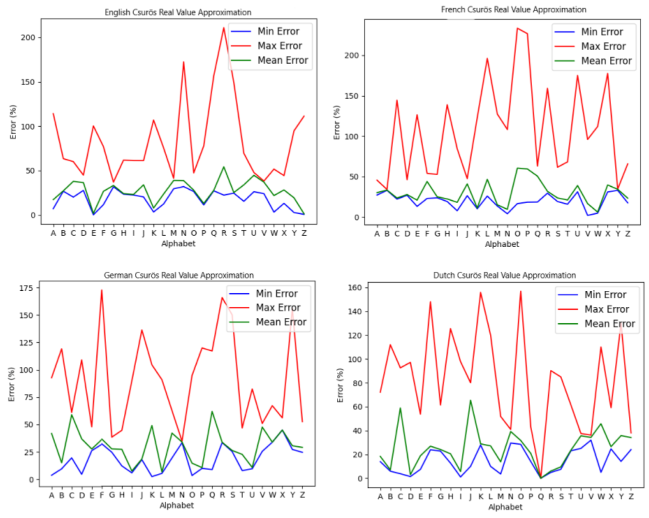


Figure 8 – Relative Error Between Exact Count and Estimated Real Value Using Csurös Counter

As visualized above, the error in some cases is unbearably high, reaching a maximum of 235% which is plain unacceptable.
Considering d=8, the results are a bit different.

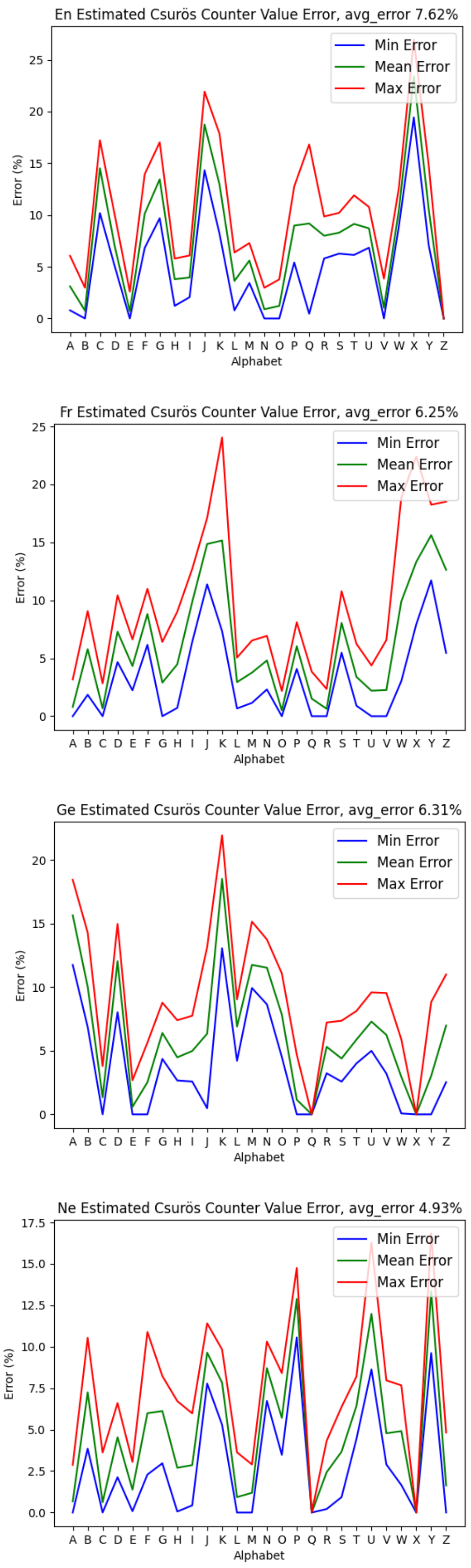
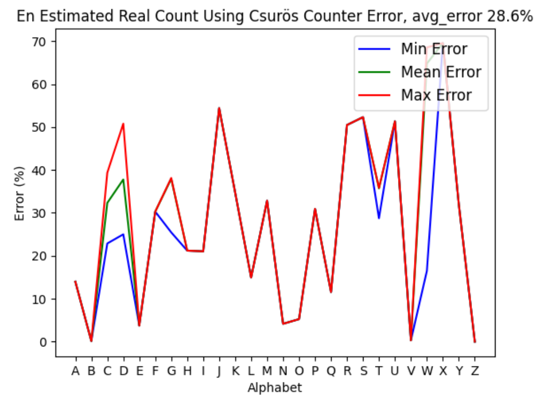
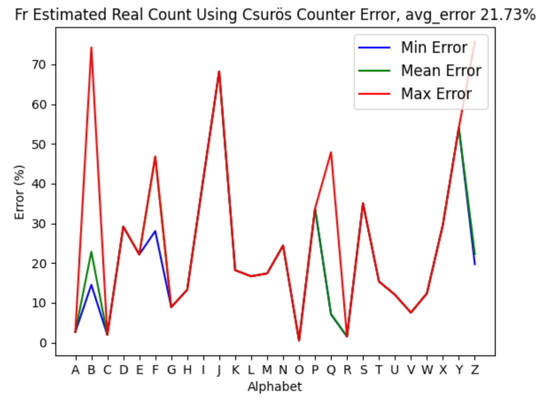
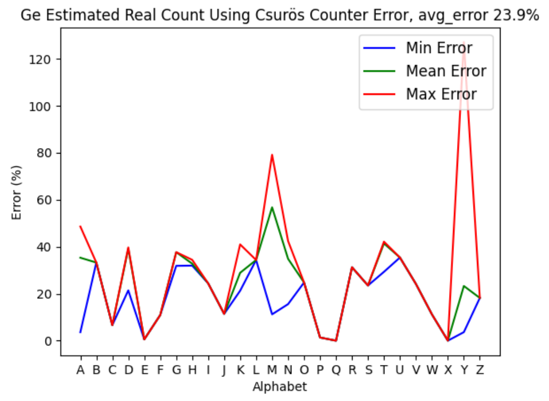


Figure 9 - Csurös Counter Value Error, d=8

As presented, the counter value error was similar to the one presented with *d = 4*. This was expected since what should change is the estimated real number of events error.







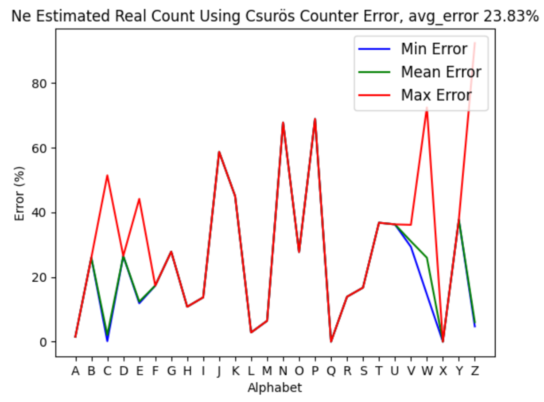
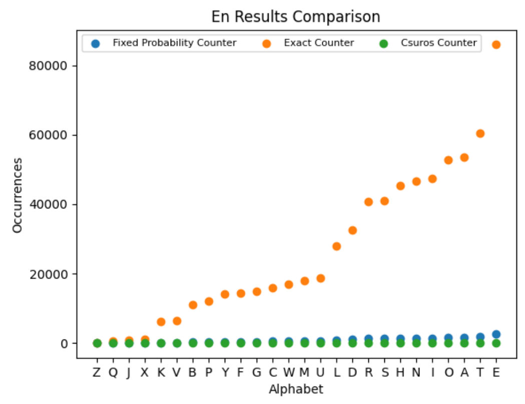


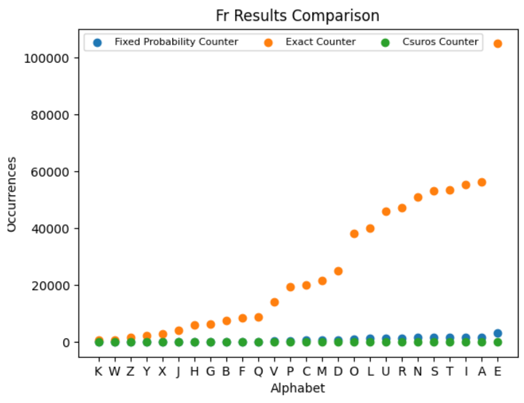
Figure 10 - Estimated Real Count Error, d=8

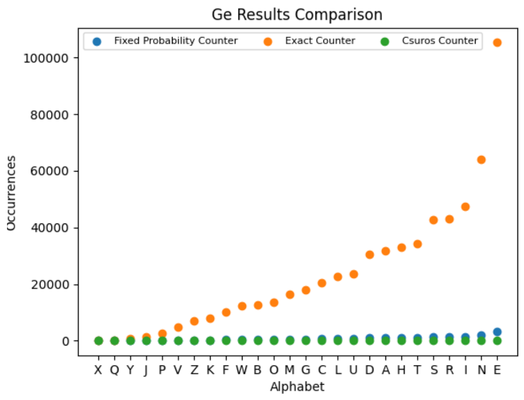
As expected, the accuracy of the counter improved. Whereas using *d = 4* reached a worst-case error above 100% multiple times, only once it occurs using *d=8*, only twice going over 80%.

A. *Final Comparison*

Considering that the number of incrementations using the Csurös counter are affected by the d factor, it is expected and visualized that the higher the *d*, the higher the counter value will be and thus higher the number of bits needed to represent a counter. However, the accuracy of the Csurös Counter depends on the *d* value, where a higher d corresponds to higher accuracy, as seen previously.







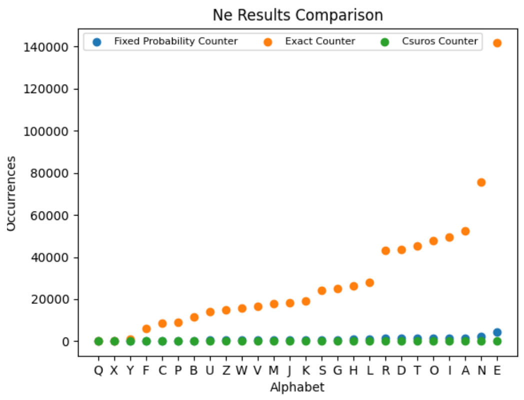


Figure 11 - Comparison Exact Counter Values, Fixed Probability Counter Values (p=1/32) vs Csurös Counter Values(d=4)

Both the fixed probability counter and the Csurös counter reduce the memory footprint well as represented in the figures above.

A. *Conclusion*

As expected, both methods showed themselves useful in reducing the memory usage of a counter. The fixed probability counter, however, is only useful for large counts, losing accuracy rapidly when the number of events is low. Furthermore, the Csurös counter is more complex and easier to manage as it offers a tradeoff between memory usage and overall accuracy. For a bigger *d* value, the accuracy will be higher but so will the memory usage.

REFERENCES

[1] R. Morris, Counting Large Numbers of Events in Small Registers, Commun. ACM, Vol. 21, N. 10, October 1978

[2] OpenGenus, "Probabilistic / Approximate Counting [Complete Overview]", (https://iq.opengenus.org/probabilistic-approximate-counting/).

[3] M. Csurös. Approximate counting with a floating-point counter. In COCOON, LNCS vol. 6196, p. 358-367, Springer, 2010