Probabilistic Counters: a comparative study between definitive counters, probabilistic counter and decreasing probabilistic counters

Pedro Miguel Rocha Oliveira

***Resumo* – Os algoritmos com contagem probablistica são usados quando é necessário contar um grande número de eventos mas não há memória disponível. São baseados em usar eventos pseudo-aleatórios para incrementar um contador.**

Neste artigo, irei comparar como um contador com uma probabilidade fixa e um contador com probabilidade descrescente com um contador normal.

***Abstract* – Approximate counting algorithms are used when needing to count a large number of events in cases where the programmer cannot use a lot of memory. They are based around using pseudo-random events to increment the counter.**

In this paper, I'll be comparing how a counter with a fixed probability, and a counter with decreasing probability fare against a regular, definitive counter.

I. INTRODUCTION

In Big Data problems, there may be a problem in dealing with memory and storage when it comes to counting events. In order to bypass this difficulty, data scientists implement an approximate counting algorithm instead of using a direct counter of events. This means that the counter won't always be incremented when the event occurs, instead being incremented when the event occurs and a pseudo-random with a certain probability happens.

This way, it's possible to count very frequent events with a smaller number, and use this small number to somewhat accurately estimate the amount of times the event occurred.

In this paper, I will be talking about two different types of approximate counting algorithms: one with a fixed probability of ¼ or 25% of happening, and one with a decreasing probability with base √2, which means as the counter grows larger and larger, the probability will decrease exponentially.

Both algorithms have their own formula to go from the saved counter to the estimated value: a fixed probability counter may estimate the value by multiplying the saved counter by the inverse of the probability, so in my case, if consider the saved value C, we can estimate the value to be around 4*C; if we consider the decreasing probability algorithm, the calculation is a bit more complex then a simple product, using the aforementioned base, and considering a saved value C, we can estimate the value to be (√2^C - √2 + 1)/(√2 -1).

As it is only an estimation, these counters will not be as accurate as a regular counter. The main of these algorithms is to sacrifice some accuracy in these results, in order to be able to save more space. As I'll show in section III, the counters will most of the times show a value from which the estimated value will be fairly close to the real value.

To test these functions, the algorithms were compared with a regularly used counter that simple increments when the event occurs, and I applied this to a problem in counting distinct words of books, with the ultimate goal being comparing the top 5 words of different translations of the same book. All the algorithms were implemented using Python3.7

II. ALGORITHMIC IMPLEMENTATIONS

The counters are fairly simple. Considering they all receive as argument the list of words to count, the three algorithms will return an hash map that maps each distinct word as a key, with the value as the corresponding counter. The differing factor for each algorithm lies in how the pseudo-random event is calculated.

A. Definitive Counter

This is the simple counter that will return the effective number of occurrences for each distinct word in the list passed as argument.

It works as an iterative loop that will go through the list, and either start the counter at for the word at 1 if the word had not appeared before, or incremement the counter.

This means there are some limitations: the algorithm may only count up to the maximum value for integers, which may not be enough for some problems. While there were no issues in this particular use case, in different problems it may not be good enough.

B. Fixed Probability Counter

This is the simpler way to implement a counter based on a probabilistic event, and will now return a value for each word from which it's possible to estimate the frequency of the words.

As the last algorithm, the algorithm goes through all elements in the list and for each occurrence, before incrementing the saved value, will launch a random number with the random function of the native libraries in Python. The counter will be incremented if and only if this random number is below 0.25.

This way, we can achieve an estimated value of around four times the maximum integer allowed to the programmer.

C. Decreasing Probabilistic Counter

The final variation of these algorithms is one that will take the base √2 to calculate the probability. As before, this will only return a number per distinct word from which it will be possible to estimate the real value of occurrences.

In yet another iterative algorithm, this will go through the list of words; however before the random event it will first calculate the probability of success based on the save counter for the word in question, using the formula 1/(√2^C) with C being the saved value. To improve on the performance, the last 10 probabilities are being saved to memory, to avoid having to make the same calculation too many times.

This algorithm allows saving values that are much larger than the previous two, as an unsigned byte may save numbers up to 5.8E38, twice more than what can be save using a 64Bit unsigned integer. However, it must be noted that a small change in the counter value results in a much different estimated value, as this is an exponential growth, as opposed to the previous two algorithms that can be described as having a linear growth.

III. TESTS AND RESULTS

With the algorithms implemented, I ran each algorithm to count the words in the book Alice in Wonderland by Lewis Carroll, in English and translated to French, German, Italian and Finnish. All books have been downloaded from the Project Guttenberg page.[1]

In these texts I filtered the punctuation and removed the stopwords for each language as was defined in the NLTK library from Python.

Then, to measure and compare the performance of each algorithm, I will be using two metrics: the mean relative error, showing the error related to the estimation from the counter, relative to how large the exact value is; and the mean accuracy ratio, showing how far the predicted value is from the real one.

To prevent the test being prone to biases or just plain bad luck, I ran these algorithms to a varying number of trials, from 10 to 10000 and kept track of the evolution of the aforementioned metrics.

Finally, as the main goal was using relatively small numbers, the metrics were calculated using the results from the regular counter as the control group.

In graph 1, it's possible to see how the trend of the mean relative error develops for the English version of the book with the top 5 words: 'like', 'one', 'little', 'alice' and 'said'.

It's possible to see that in the beginning the error is still erratic, but the more trials there are, the values tend to stabilize. And after this stabilization, we see that the relative error from the estimation based on the decreasing probability is actually two to three times higher than the correspondent with the fixed probability counter.

Now considering the mean accuracy rate, which shows just how far, in average, the estimated value is from the real value, we see that at the end of many trials, all values start to stabilize towards being perfectly accurate. However, the decereasing probability counter shows the most erratic evolution, reaching averages much higher or much lower than the fixed probability with a small number of trials.

As can be seen in figures 3 through 10, these conclusions are consistent throughout all languages tested for, with their respective top 5 most common words.

These results are around what was to be expected. As said in the previous section, a small change in the counter results in a much larger difference in estimated value for the decreasing probability counter than for the fixed probability counter. Since these are all made with pseudo-random events, these differences in counters are bound to happen.

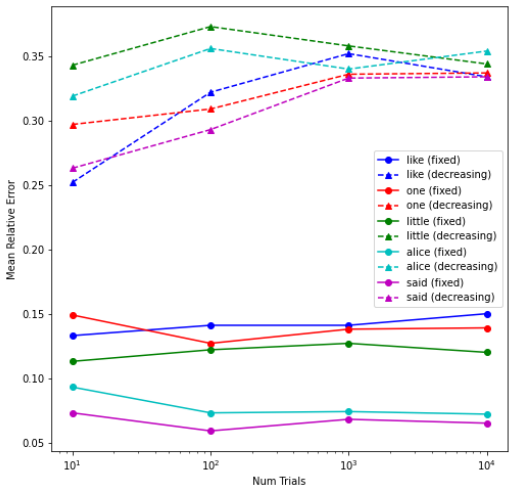

Fig. 1 - Graph relating the number of trials with the mean relative error using the English book.

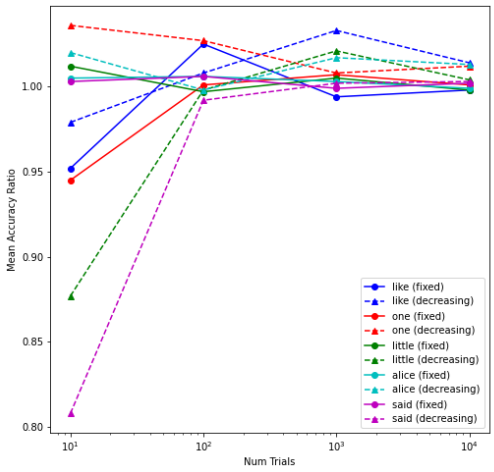

Fig. 2 - Graph relating the number of trials with the mean accuracy ratio using the English book

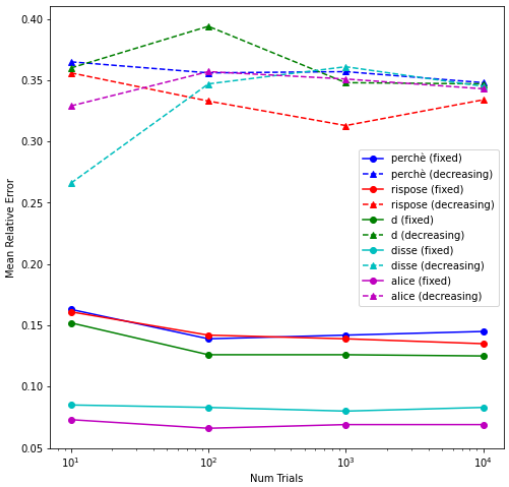

Fig. 3 - Graph relating the number of trials with the mean relative error using the Italian book.

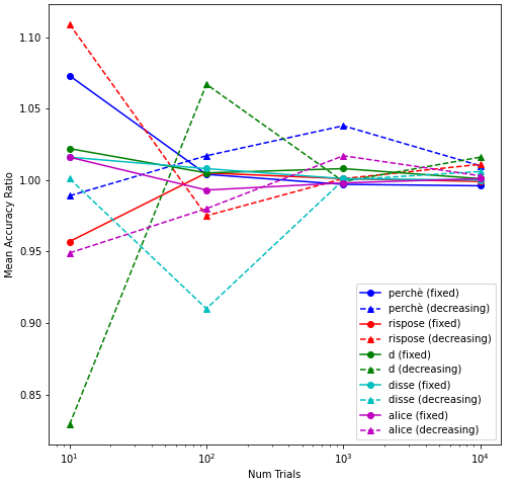

Fig. 4 - Graph relating the number of trials with the mean accuracy ratio using the Italian book

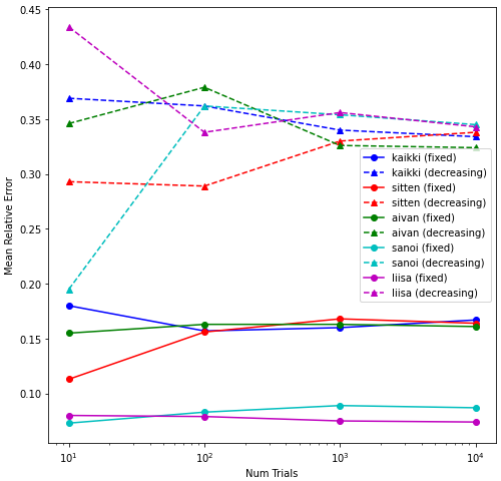

Fig. 5 - Graph relating the number of trials with the mean relative error using the Finnish book.

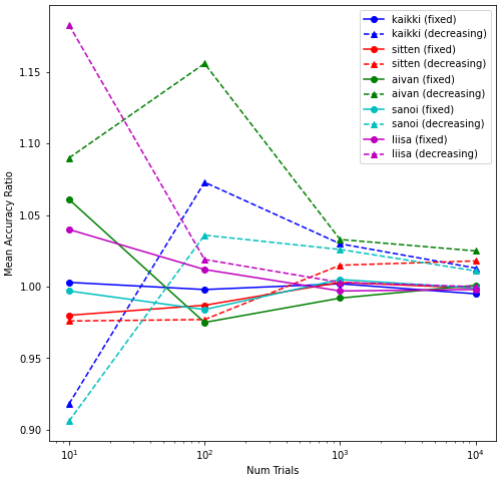

Fig. 6 - Graph relating the number of trials with the mean accuracy ratio using the Finnish book

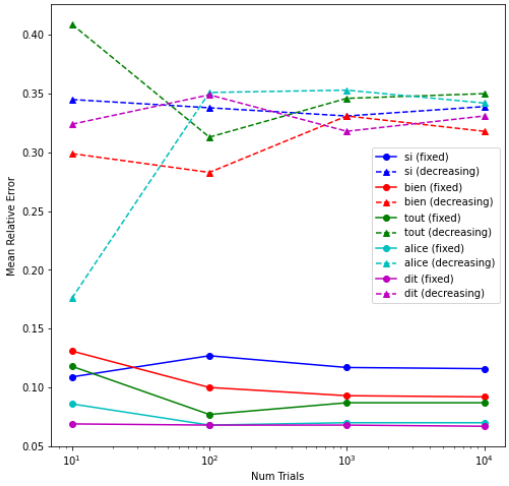

Fig. 7 - Graph relating the number of trials with the mean relative error using the French book.

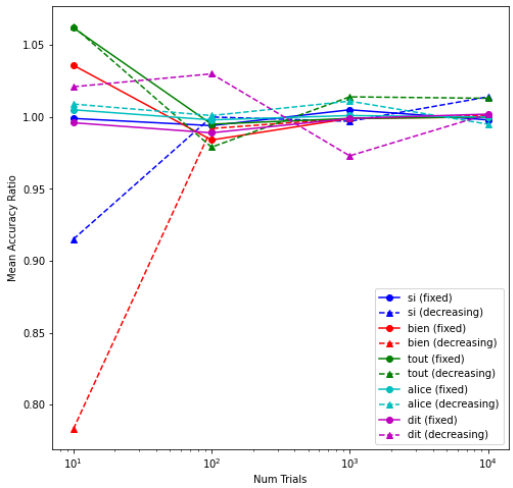

Fig. 8 - Graph relating the number of trials with the mean accuracy ratio using the French book

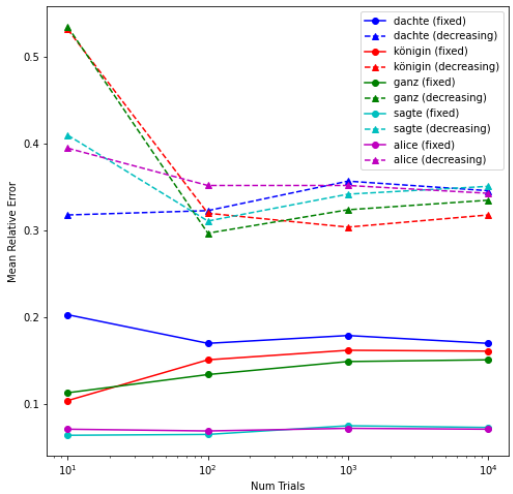

Fig. 9 - Graph relating the number of trials with the mean relative error using the German book.

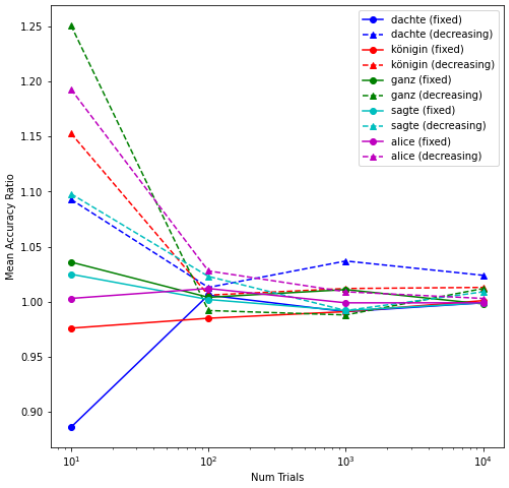

Fig. 10 - Graph relating the number of trials with the mean accuracy ratio using the German book

IV. CONCLUSION

With this project, it's possible to see the strengths and weaknesses of each implemented counter.

An algorithm using fixed probability will have results that are much closer to reality than its decreasing probability counterpart, which, smaller data, may be ideal. In this case, counting words in a book, in this case a relatively small children's book, would mean that the values never grow too large. Of course, since this was to study the effectiveness of each counter, this had to be the case, as the problem in question would have to be solved by a deterministic counter as well, as the control group.

However, there are some problems with the fixed probability counter that must be noted. The limit of a fixed probability counter won't actually be too different from the limit of a regular counter, unless we reduce the probability to count by a lot, which may in itself result in other problems. So, in cases where storage space if a big issue and with numbers that are really large, and if some discrepancy is allowed, than the decreasing probability conuter seems to be a better solution, since, even though it causes a large irregularity and as a larger relative error rate, it can store much larger amounts to data.

In this project, the word that appeared most often was "dit" from the French edition of the book, with a total of 416 occurrences. The fixed probability counter counted it as 104, making it possible to write using a single byte. Even more impressively was the decreasing probability algorithm that counted it as 14, which makes it possible to fit in only 4 bits, proving that this one is the best for saving memory space.

REFERENCES

[1] https://www.gutenberg.org