On the usage of approximate counters for the occurrences of words

Manuel Gomes, 88939

***Abstract* –This report present four algorithms to solve a minimum weight dominating set graph problem. The first one is an exhaustive search algorithm, while the rest are based on greedy heuristics. A formal analysis is presented for each algorithm. After their implementation in Python 3, an experimental analysis of each algorithm is described.**

***Resumo* –Este relatório apresenta quatro algoritmos com o objetivo resolver o problema de grafos "minimum weight dominating set". O primeiro é um algoritmo de procura extensiva, enquanto que o resto são baseados em heurísticas vorazes. Uma análise formal é apresentada para cada algoritmo. Após a sua implementação em Python 3, uma análise experimental de cada algoritmo é descrita.**

I Introduction

The need for gathering statistics on a large number of events is a very common occurrence in data science applications, such as social media networks and search engines. However, the resources are finite and the totality of data to analyse is larger than the available memory. The largest challenge in this area is to store the data in a storage efficient way. One solution for this problem is the usage of probabilist counter algorithms. In these algorithms, an occurrence is not always accounted, being dependent on a probabilty. Although this reduces the memory used, the accuracy is reduced as well.

The objective behind this report is to explore the accuracy and efficiency behind some probabilistic counters in counting the occurrence of words inside a book. The report is divided in five section: the first (section I), where the problem is introduced; the second (section II), where the algorithms used are described; the third (section III), where the experiments are described and results for them are detailed; and the fourth (section IV), where conclusions are extracted.

II Algorithms Used

To count the number of occurrences of every word in a book, an exact counter and two different probabilistic counters were used. The two different probabilist counters were a fixed probability counter and a Csűrös' counter.

II-A Exact Counter Algorithm

An exact counter is a data structure that maintains an accurate count of unique items in a stream of data. In this specific case, the counter maintains an accurate count of words inside a book.

In this algorithm, an empty data structure is created. This data structure creates a subdivision for every new word accounted for. Inside that subdivision, an integer keeps count of every account of that specific word. This can be better visualised in Algorithm 1.

Algorithm 1 Exact counter algorithm

```
Inputs:
    book ← string of text to be analysed
Initialize:
    counter ← data structure used for
    counting the occurrences of every word
for word in book do
    if word in counter then
        counter[word] += 1
    else
        counter[word] ← data structure
    end if
end for
```

This algorithm allows for an exact count of unique items, with no error margin. The downside of using an exact counter is that it can use a lot of memory.

II-B Fixed probabilistic counter

A fixed probabilistic counter is a data structure that keeps count of the number of distinct elements in a stream of data, with a certain level of precision. It employs random sampling to determine the count and the degree of accuracy can be altered by altering the probability of sampling. This probabilty remains constant for the entire duration of the algorithm.

Similarly to Algorithm 1, an empty data structure is created. Every time a new word is analysed, a random number from 0 to 1 is generated. If this number is lower than the sampling probability, then the word is sampled. If a subdivision for the sampled word does not exist inside the data structure, this division is created. Inside that subdivision, an integer keeps count of every account of that specific sampled word. This algorithm is explained in Algorithm 2.

Algorithm 2 Fixed probability counter algorithm

Inputs:
book ← string of text to be analysed
p ← sampling probability
Initialize:
counter ← data structure used for counting the occurrences of every word
RandomNumber ← function that generates a number between 0 and 1
for *word* in *book* **do** $r_{number} = RandomNumber$
if $r_{number} < p$ **then**
if *word* in *counter* **then**
$counter[word] += 1$
else
$counter[word]$ ← data structure
end if
end if
end for

II-C Csűrös' probabilty counter algorithm

Another type of probabilistic counter are the counters with decreasing probabilty. A decreasing probabilistic counter is, similarly to a fixed probabilistic counter, a data structure that keeps count of the number of distinct elements in a stream of data, with a certain level of precision. It also employs random sampling to determine the count. However, the sampling probabilty decreases as the algorithm evolves. This allows for an accurate count in the first occurrences of an event without using to much memory.

One example of this type of counter is the Csűrös' counter. This is a floating-point counter defined with the aid of a design parameter $M = 2^d$, where d is a nonnegative integer. This counter counts with deterministic updates for the first M occurrence, similarly to Algorithm 1. The next updates are probabilistic, with the next M updates having a probability of $\frac{1}{2}$, followed by M updates with probabilty of $\frac{1}{4}$, etc... This algorithm is described in Algorithm 3 and Figure 1.

Algorithm 3 Csűrös' counter algorithm

Inputs:
book ← string of text to be analysed
d ← design parameter
Initialize:
counter ← data structure used for counting the occurrences of every word
RandomNumber ← function that generates a number between 0 and 1
for *word* in *book* **do** $r_{number} = RandomNumber$
if $r_{number} < (\frac{1}{2})^{truncate(counter[word]/2^d)}$ **then**
if *word* in *counter* **then**
$counter[word] += 1$
else
$counter[word]$ ← data structure
end if
end if
end for

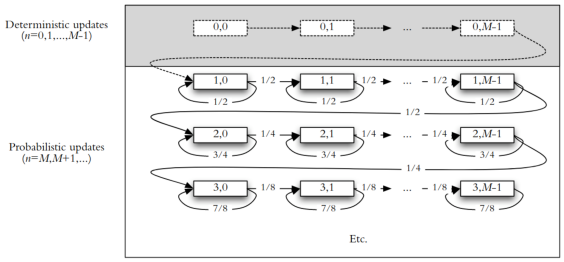

Fig. 1
CSŰRÖS' COUNTER EXEMPLIFIED IN GRAPHICAL FORM.

III RESULTS

IV CONCLUSIONS