Birthday Paradox

Davide Di Mauro
Politecnico di Torino
Student ID: s306089
s306089@studenti.polito.it

***Abstract*—The objective of this homework is to devise an antiplagiarism software specialized on poems, specifically on "La Divina Commedia" by Dante Alighieri.**

I. Preliminaries

Starting from a raw text file some preprocessing steps were applied in order to remove punctuation, repeated blank spaces, uppercase letter and newline characters. After the preprocessing step the text is divided in 6 words sentences with an average size per sentence of **90.93 B** (Table I).

TABLE I: Text Preprocessing

	# words	# verses	# sentences (6 words)
Original:	96858	14753	n/a
After preprocessing:	96730	14342	96700

II. Set of sentences

A python set containing all the 96700 distinct sentences is created as a baseline. The effective memory occupancy of this data structure is **12683 kB**. With this solution the time to search for an element is $O(N)$ where N is the number of elements of the set, and of course there is no probability of false positives.

III. Fingerprint set

This approach consists in generating a fingerprint $h(x)$ computing an hash for each sentence and store them in a set, with access time $O(N)$. The probability of having false positives in this case is a function (1) of the number of element to store m and the range of each fingerprint $n = 2^b$.

$$Pr(false\_positive) = 1 - \left(1 - \frac{1}{n}\right)^m \tag{1}$$

Using Equation 2 derived firm the generalized version of the birthday problem is also possible to compute analytically the number of bits necessary to get a probability p (in our case $p = 0.5$) of having a collision when storing all the sentences in a fingerprint set.

$$n = -\frac{m^2}{2\log(1-p)} \tag{2}$$

Figure 1 shows the experimental number of collision in function of the number of bits. In our experiment we obtain no collisions using **32 bits** fingerprints (**7118 kB** of memory occupancy) while theoretically we computed that 33 bits where necessary to get a probability 0.5 of fingerprint collision. This proves that the theoretical approximation is very precise.

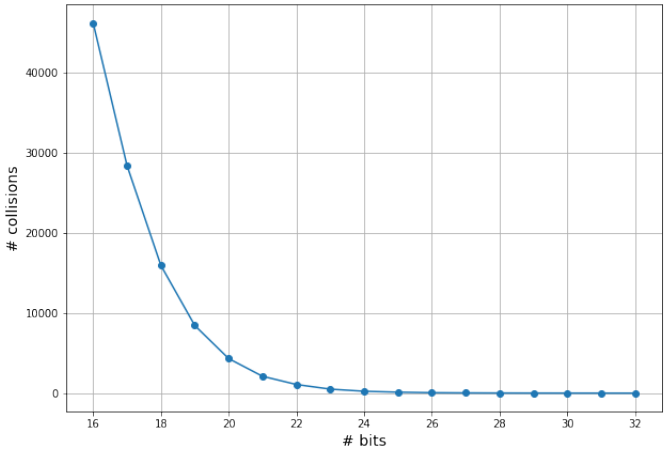

Fig. 1: # of collisions the fingerprint set

IV. Bit string array

Another technique analyzed is the bit string array, where a fingerprint $h(x)$ is computed for each sentence and then the value of the bit string array in position $h(x)$ is set to 1. The access time of this data structure is $O(1)$ with the trade-off of utilizing more memory with the same size b for the fingerprints, compared to the set of fingerprints. A comparison of the theoretical and experimental $Pr(false\_positives)$ is provided in Figure 2 where we can see that the theoretical probability (Equation 1) is basically identical to the empirical one (Equation 3).

$$Pr(false\_positives) = \frac{\#\_values\_set\_to\_1}{n} \tag{3}$$

V. Bloom filters

The last method analyzed is the bloom filter, that works exactly like the bit string array, but where multiple k hash functions are computed for each element to store. To check if an element is stored, all k hash functions are computed and all the elements in position h_1 ... h_k must be set to 1. To generate different hashes we simply modify the input x appending to

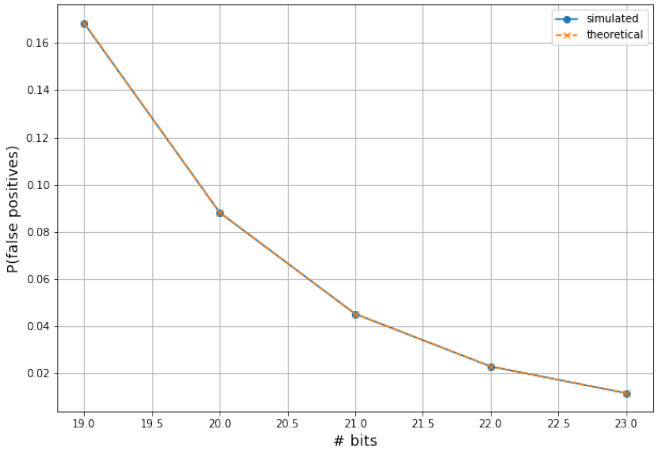

Fig. 2: $Pr(false\_positives)$ of the bit string array

the string an additional character so that $h_1 = h(x)$, $h_2 = h(x+1)$, ... , $h_k = h(x+k-1)$.

The first thing to do when using bloom filters is to compute the optimal value of k that minimize the probability of false positives, using Equation 4. In Figure3 we show how the optimal number of hash functions increases exponentially with the number of bits.

$$k_{opt} = \frac{n}{m} \log(2) \tag{4}$$

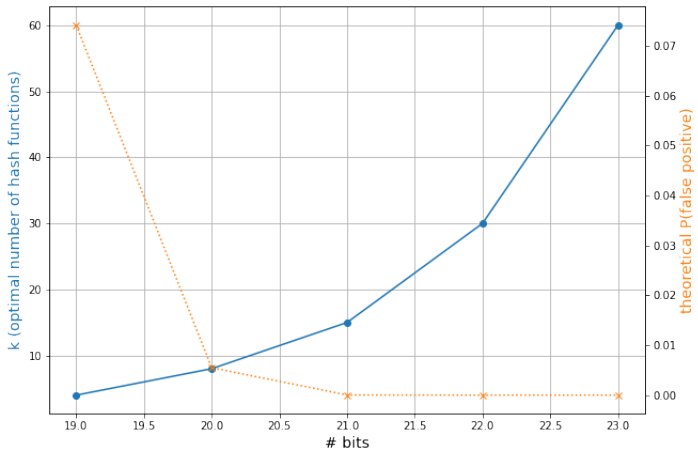

Fig. 3: Optimal number of hash function

We then proceed to simulate the probability of false positives (Equation 3) and compare it with the theoretical one (Figure 4), obtained using the following equation (5):

$$Pr(false\_positive) = \left(1 - \left(1 - \frac{1}{n}\right)^{mk}\right)^{k} \tag{5}$$

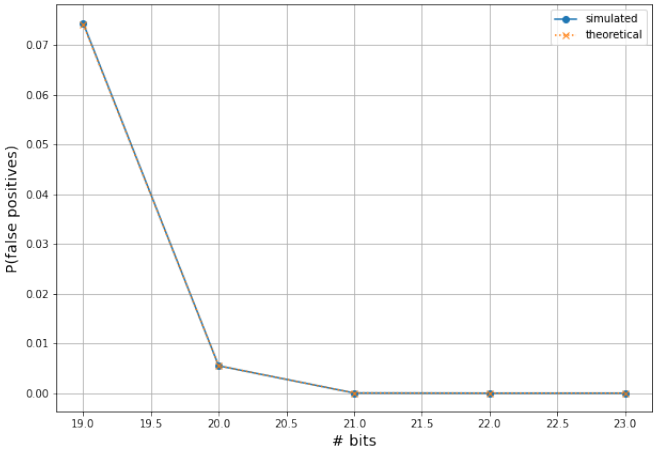

Fig. 4: $Pr(false\_positives)$ of the bloom filter

Finally in Figure 5 we plot the difference between the estimated value of stored element and the actual one, in order to show how accurate is the formula to estimate the number of distinct elements stored in a bloom filter with n bits and k hash functions (Equation 6)

$$distinct\_elements = -\frac{n}{k} \ln\left(1 - \frac{N}{n}\right) \tag{6}$$

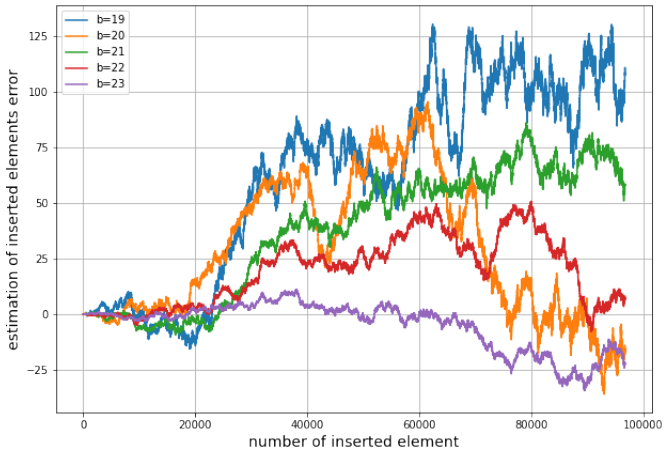

Fig. 5: Optimal number of hash function

VI. Conclusions

In conclusion we summarize the results obtained in Table II and Figure 6. We noticed that each of these techniques has its own advantages and disadvantages:

- The set of fingerprints reduces almost by a factor 2 the memory occupancy compared to storing the sentences, while also having a significantly low probability of false

positives. The only trade-off is that this solution doesn't scale well, and the $O(N)$ access time.

- Bit array and bloom filters solve the problem of the access time, at the cost of memory and probability of false positives. Actually bloom filters reduce drastically the number of false positives with the same amount of memory occupancy, at the cost of having to compute multiple hash functions.

For our specific use-case, the following assumptions were made:

1) Usually poems are very small compared to "La Divina Commedia", thus access time is not a big concern
2) We want to have a relatively precise system, so we required to have $P(false\_positives) \leq 10^{-4}$

Under those assumptions we would choose the set of fingerprints, but changing the aforementioned assumptions to the user specific needs we think that also both the solutions using bloom filters with $b = 20$ and $b = 21$ can result in a more optimal solution.

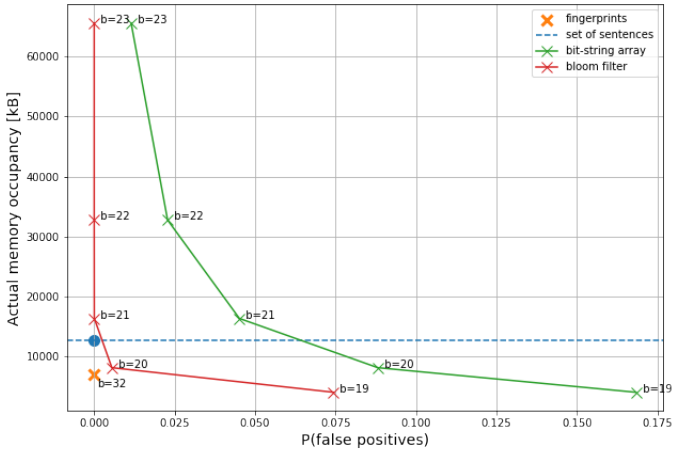

Fig. 6: Memory occupancy comparison

TABLE II: Summary

Data structure	**Memory [kB]**	**Prob. false positive**
Set of sentences	12683	*N/A*
Fingerprint set	**{b=32} 7118**	**2.51e-05**
Bit string array	{b=19} 4096, {b=20} 8192, {b=21} 16384, {b=22} 32768, {b=23} 65536	0.168, 0.088, 0.045, 0.023, 0.011
Bloom filter	{b=19} 4096, **{b=20} 8192**, **{b=21} 16384**, {b=22} 32768, {b=23} 65536	0.074, **0.005**, **3.00e-05**, 8.92e-10, 7.85e-19