Lab 10: Bit-String Hashing

Tommaso Massaglia
Politecnico di Torino ID: s292988
s292988@studenti.polito.it

I. Introduction

The goal of the two labs is to simulate and analyze the performance of bit-string hashing, comparing it with the performance of an equivalent fingerprint set.

II. Assumptions

The following assumptions are made:

- A dataset of words or sentences is considered for each simulation, each dataset is composed of m items
- Each simulation is ran for b bits
- The total number of keys is n and is computed as 2^b
- The theoretical probability of false positives for bit-string storage is given by:

$$p_{falsepositive} = \frac{\sum_{bit\_string} \text{"1"} bits}{2^b} \tag{1}$$

- The theoretical probability of false positives for a fingerprint set of b bits for a dataset of m elements is given by:

$$p_{falsepositive} = 1 - (1 - \frac{1}{2^b})^m \tag{2}$$

- As every *char* takes one byte to store, the memory occupancy of saving just the strings (in MB) is given by:

$$size_{strings} = \frac{\sum_{word} \sum_{char} 1}{8 \cdot 1024^2} MB \tag{3}$$

- The memory required to store the bit-string hash is given by:

$$size_{bit-string} = \frac{2^b}{8 \cdot 1024^2} MB \tag{4}$$

- The memory required to store a fingerprint set is given by:

$$size_{fingerprint} = \frac{m \cdot b}{8 \cdot 1024^2} MB \tag{5}$$

III. Input Parameters

Given the assumptions, the following input parameters are considered:

- An hashable dataset: for the simulations considered a dataset of 660000 Italian word was taken from here and a list of over 30M movie titles (which is filtered only for Italian movies down to 3M titles) from here were used
- b: the number of bits we want to run the simulation for (from 19 to 26 in our case)

IV. Output Metrics

The following metrics were used:

- The total memory occupancy (eq.3, 4 and 5)
- The probability of false positives (eq. 1 and 2)
- The Mean Absolute Error:

$$MAE = \frac{|bit\_string\_res - fingerprinting\_res|}{\#experiments} \tag{6}$$

V. Data structure and Algorithms

Initially, the words are stored in a *pandas* dataframe for ease of importing; then, to compute the bit-string, a *numpy* array of length 2^b is initialized with $int8$ zeros, to minimize as much as possible its size (as we only need to store 1s and 0s inside).

When the simulation is run for b bits, at each step we compute the b hash for a word in the word list and change the bit in the corresponding position in the bit-string to a 1.

What allows us to run the simulation as fast as we do is mainly eq. 1: this formula, used to find false positives, is the equivalent of generating a word whose hash is equal to each possible bit position in the bit string (so words with hashes from 0 to $2^b - 1$), or directly the hash linked to it, but requires no generation and no confidence interval computation.

During the lab, I tried to test an equally sized dataset with mostly different elements with respect to the considered one (taken from here), but since little to no difference was shown in the results (a MAE of 0.013), the proposed formula was adopted.

VI. Results

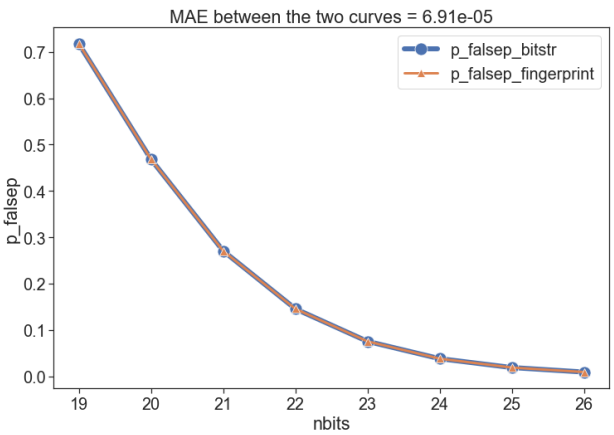

Fig. 1. Probability of false positives

Figure 1 shows the difference in probability of false positives when comparing bit-string hashing with fingerprinting (using eq. 1 and 2); as shown visually as well as by the MAE, the two curves are practically identical, highlighting no difference in performance when considering this metric.

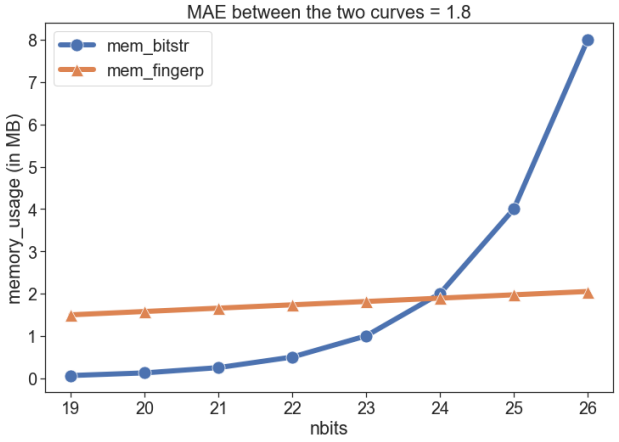

Fig. 2. Memory usage

Starting from a baseline of $7.23MB$ (computed using eq. 3), figure 2 shows the difference in memory usage when using the two types of hashing (memory computed using eq. 4 and 5).

A. *optional: results on the movies dataset*

The size of the dataset is 3.5M elements, the baseline memory occupancy is $64.42MB$

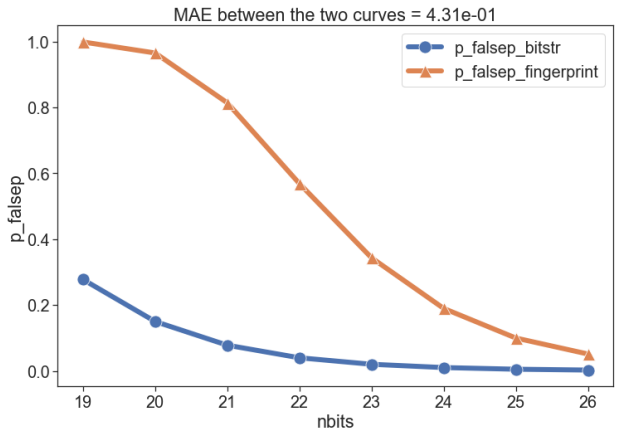

Fig. 3. Probability of false positives

The results observed in fig. 1 do not hold for the movie titles dataset, with bit-string hashing outperforming fingerprinting by a large margin; as the bit count increases though, we can see that the two curves tend to converge, suggesting that eventually we would see a similar result.

Similarly, bit-string hashing performs better (as it's less dependent on the number of elements stored) for low bit counts, but due to the shape of its curve and the nature of the hashing method its bound to be worse as the bit count increases.

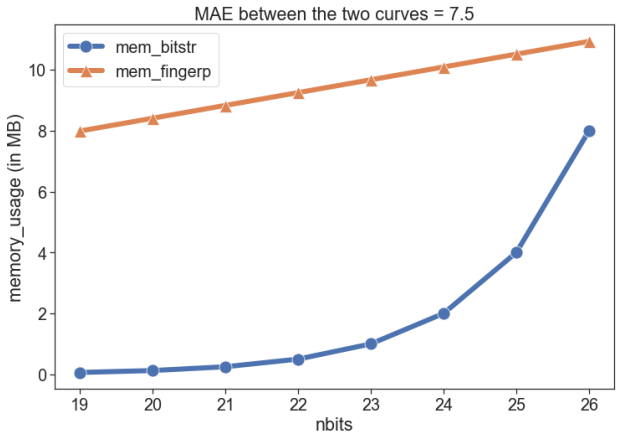

Fig. 4. Memory usage

VII. Discussion

Since there was practically no difference in $p_{falsepositive}$ for the words dataset, the discussion will focus on the memory aspect of the two hashing methods.

As we can infer from figure 2, with lower values of b, bit-string hashing outperforms fingerprinting, and this trend inverses as we go over 24 bits. When considering the baseline as well, we can say that for low b bit-string hashing is more convenient, as we increase b, fingerprinting becomes the better solution with no performance loss; bit-string hashing becomes worse than the baseline for higher bit counts (probably due to an oversizing of the bit-string which for $b = 29$ allows for 536870912 hashes, over $1000x$ the number of words we need to store). Fingerprinting on the contrary is less dependent on the precision we want to reach and more on the number of words we want to store, thus becoming less efficient for lower b but more as the b increases.

This is further highlighted when considering the movies dataset. The same conclusions can be reached as the probability of false positives converges as b becomes reasonable for the size of the dataset; memory usage for bit-string hashing increases rapidly as b increases and remains mostly stable for fingerprinting (still being way lower than direct string storing as movie titles are longer than simple words thus providing better gains out of hashing).

Wrapping up, we can say that there will always be a point where fingerprinting outperforms bit-string hashing, and that point is dependent on the number of elements we want to store and on the number of bits we want to use to store them.