Computer aided simulations and performance evaluation - Lab 3

Ruben Berteletti - s277757

February 2021

1 Introduction

The purpose of this laboratory is, given a file with the English words list containing all the 370103 words (available at `https://github.com/dwy1/english-words/blob/master/words_alpha.txt`), to evaluate the performances of *words-set*, *fingerprint-set*, *bit string array* and *bloom filter* in the set-membership problem.
To get a precise measure of the memory occupancy, the *Pympler* library has been used.

2 Fingerprinting

In order to evaluate the performances for the fingerprint set, the simulator takes as input only the file with the 370103 English words; no other variables are required since in this case the approach is completely deterministic.
The output metrics are:

- b^{exp}: the minimum number of bits that guarantee that 0 collisions are experienced storing the words in a fingerprint set;
- b^{theo}: the number of bits necessary to get a probability 0.5 of collision storing the words;
- The *minimum* amount of memory required for both the words set and the b^{exp}-fingerprint set;
- The *actual* amount of memory required for both the words set and the b^{exp}-fingerprint set;
- The false positive probability for the b^{exp}-fingerprint set.

To achieve those results only a couple of python set are required: one for the words set and one for the fingerprint set.
The simulative approach is to run a loop over the number of bits allowed for fingerprint and step-by-step inserting a new word in the set while no conflict are experienced. A conflict implies then, that the number of bits reserved for the fingerprint are not enough to store all the words. In this setting the simulation ends when the size of the fingerprint set is equal to the total number of words.
This leads to a $b^{exp} = 39$. b^{theo} instead is computed as $b^{theo} = \log_2 \frac{w}{\epsilon}$ where w is the total number of words and ϵ is the probability of conflict i.e. 0.5; leading to $b^{theo} = 19$. The relation between b^{exp} and b^{theo} can be seen as $b^{exp} = \lim_{\epsilon \to 0} b^{theo}$.
Regarding the sizes two ways have been adopted: for the actual size the measure has been obtained with the *Pympler* method `asize.asizeof()` while for the theoretical memory required, given the average number of characters in the English words which is 4.7 and considering the equation 1 char = 1 byte, it has been computed as $theomemory = totwords * 4.7$.
Finally the probability of false positive can be computed as $P(FP) = 1 - (1 - \frac{1}{n})^w$ where w is the number of words and $n = 2^{b^{exp}}$, which with $b^{exp} = 39$ leads as result 0.000067%.
The previous results along with the memory occupancy are summarized in the following table:

Storage	Bits per fingerprint	Prob. false positive	Min theoretical memory	Actual memory
Word set	N.A.	N.A.	1698.71 KB	21973.55 KB
Fingerprint set	39	$6.73 * 10^{-5}$%	1761.96 KB	16860.96 KB

Table 1: Summary of the simulation's result.

Note that since all the parameters are fixed, here the simulation is completely deterministic i.e., no confidence intervals are required.

3 Bit string hashing

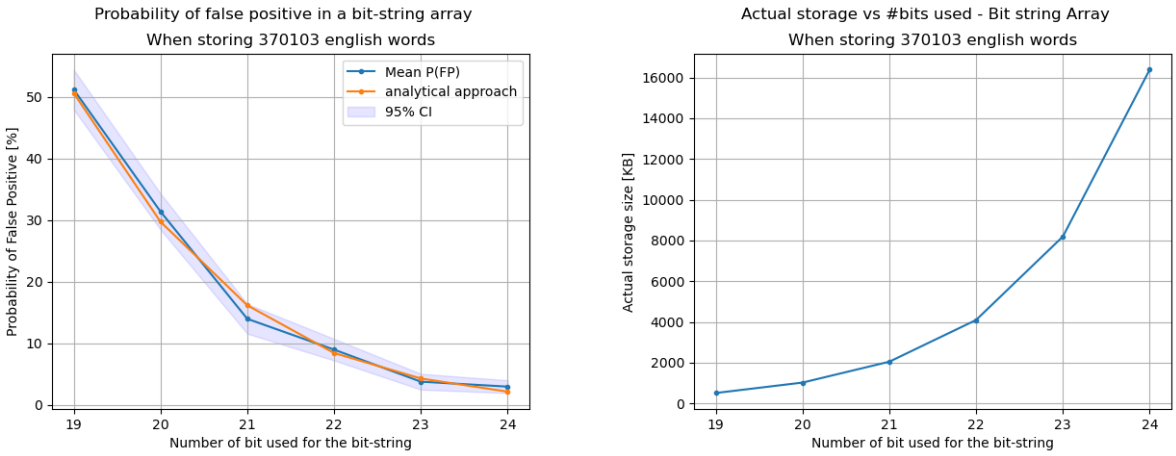
In this case there are more inputs parameters, since in order to test the performances is required to generate some fake hashes and try to put them in the bit string array. So the inputs are:

- The number of bits reserved for the fingerprint: $b = [19, 20, 21, 22, 23, 24]$;
- The file containing the English words;
- The number of runs and the confidence level, respectively set to 100 and 95%;
- The number of attempt the simulator tries to generate fake hashes set to 10;
- The seed, for replicability.

Whereas the outputs metrics are for each b:

- The theoretical and simulative probability of false positive;
- The amount of theoretical storage and actual storage required.

The main data structures used in this simulation are Python lists aimed to track storage and probability for each number of bit and a boolean *Numpy* array of length 2^b where b is the number of bits reserved for the fingerprint.
The approach used is close to the one adopted for the fingerprint set, but to test by simulation the probability of false positive, a fake hash is generated trough `random.randint(0, 2**n_bits-1)` simply exploiting the fact that the hash function result is an integer in the range $[0, 2^b - 1]$.
In the following chart we can appreciate the simulation outputs:



(a) (b)

Figure 1: Bit string array output charts. In figure (a), the probability of false positive (conflicts) in function of the number of bits used to store the fingerprint in a bit string array; analytical and simulative approach (95% CI).
In figure (b) the actual storage in KB in function of the number of bits used for storing the fingerprint.

Regarding the probability of false positive, as expected, the more bits are allowed to store the words' fingerprint, the less in the probability of false positive; starting with a probability near to 50% with 19 bits and coming to a probability really close to 0 with 24 bits. Moreover, the analytical line is always within the 95% confidence interval confirming the goodness of the simulation.
Looking instead at the actual storage the more bits are used and the more the size increases.
Both curves follow an exponential trend (inversely for the probability of false positive) and this is justified with the fact that the key role is played by 2^b wher b is the number of bits used; in fact by the theory: $P(FP) \propto 2^{-b}$ and $Storage \propto 2^b$.
All the result are summarized in the table below:

Bits per fingerprint	Prob. false positive [%]	Min theoretical memory [KB]	Actual memory [KB]
19	51.20	64	512.05
20	31.40	128	1024.05
21	14.00	256	2048.05
22	9.00	512	4096.05
23	3.80	1024	8192.05
24	3.00	2048	16384.05

Table 2: Bit string array: Summary of the simulation's result.

Comparing Table 2 with Table 1 we can get that the bit string array compared to the fingerprint set and the words set, has the capability to reduce the memory required depending on the bits used for the fingerprint (at least the theoretical memory, not the actual due to the Python internal management) at the cost of a percentage of false positive allowed. This, depending on the application can lead to an huge improvement on the performances.

4 Bloom filter

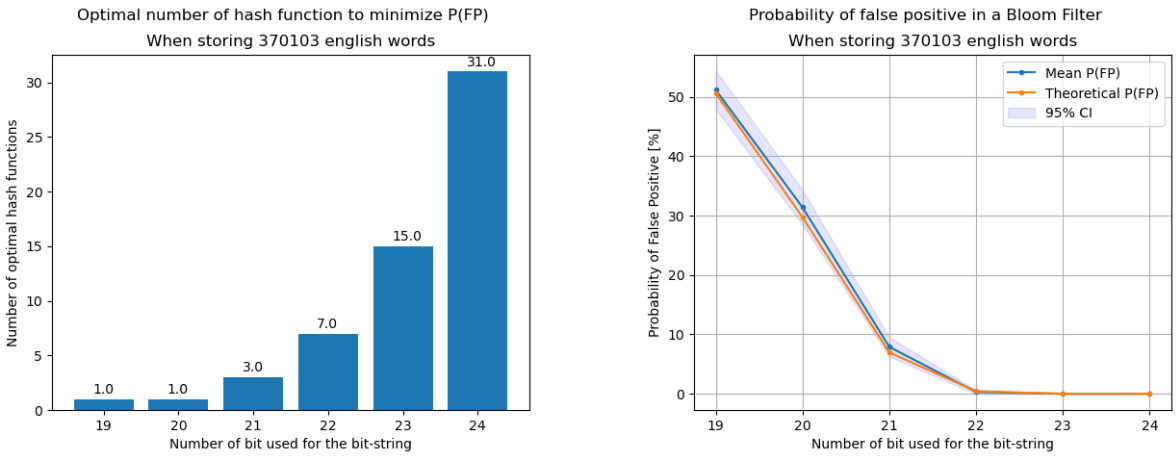
The bloom filter simulation engine shares with the bit string array most of the code, in fact we can see the bloom filter as an improvement of the bit string array where multiple hash function are used instead of just one. So, the input parameters are:

- The number of bits reserved for the fingerprint: $b = [19, 20, 21, 22, 23, 24]$;
- The file containing the English words;
- The number of runs and the confidence level, respectively set to 100 and 95%;
- The number of attempt the simulator tries to generate fake hashes set to 10;
- The seed, for replicability.

The output metrics are slightly more, since the analysis required is deeper. For each b:

- The theoretical and simulative probability of false positive;
- The amount of theoretical storage and actual storage required;
- The optimal number of hash functions k^{opt} that minimize the probability of false positive;
- The probability of false positive in function of the number of the hashes (range $1 \rightarrow 32$, optional);

The data structures used in the simulation are Python lists to track k^{opt}, probability and storage through the number of bits for the fingerprint; a boolean *Numpy* array of length 2^b where b is the number of bits reserved for the fingerprint and a Python dictionary (key = number of bits; value = lists of false positive probabilities) for measuring the optimal number of hash function (the one which minimize the probability of false positive). Once the simulator gets the number of optimal hash functions k^{opt}, the algorithm is the same as the bit string one, but instead of a single hash function, k^{opt} are used. Keeping in mind the previous sentence, is clear that in case of $k^{opt} = 1$ the Bloom filter and the bit string array are exactly the same data structure with the very same performances. This is highlighted in Figure 2, where for 19 and 20 bits reserved for the fingerprint $k^{opt} = 1$ and the probability of false positive is exactly the same.
The Bloom filter power arises when $k^{opt} > 1$ where the probability of false positive is drastically reduced, in particular with the simulation setting is very close to zero for $b > 21$.
Also here, the probability of false positive 95% confidence interval keeps inside the theoretical probability, which again it guarantees the goodness of the model.
Regarding the storage instead, since the starting data structure is the same and the algorithm is very similar, both theoretical and measured size are equal to those of bit string array.



(a) (b)

Figure 2: Bloom filter output charts. In figure (a), the number of optimal hash functions in function of the number of bits used to store the fingerprint;
In figure (b) the probability of false positive in function of the number of bits used for storing the fingerprint analytical and simulative approach (95% CI).

Also for bloom filter the result are summarized in the following table:

Bits per fingerprint	Optimal number of hash	Prob. false positive [%]	Min theoretical memory [KB]	Actual memory [KB]
19	1	51.20	64	512.05
20	1	31.40	128	1024.05
21	3	7.90	256	2048.05
22	7	0.30	512	4096.05
23	15	≈ 0	1024	8192.05
24	31	≈ 0	2048	16384.05

Table 3: Bloom filter: Summary of the simulation's result.

Given these results is clear that the bloom filter is more efficient than the bit string array in handling the probability of false positive, in fact for $b \geq 21$ the false positives are more rare.
Comparing the three data structures aimed to solve the *set-membership problem* i.e. fingerprint set, bit string array and bloom filter, the most efficient one is for sure the bloom filter, that at the same storage required, it allows to have much less false positives.
The difference between data structures could be more appreciable in case of bounded memory: having 1MB (that means 8388608 bit and then $\log_2 8388608 = 23$ bit reserved for the fingerprint) of available memory for the storage the performances are:

Storage [KB]	Probability of false positive [%]
Words set	It stores only 19660 words
Fingerprint set	8.4458
Bit string array	4.3161
Bloom filter	0.0019

Table 4: Comparison of data structures' performances in the *set-membership problem* having 1MB of storage available. Note that for the *approximated set-membership problem* i.e., in case of fingerprint, this setting is the same as having 23 bits reserved for the fingerprint.

We can conclude that introducing the approximated set membership problem, then allowing some conflicts, the saving in term of storage is huge: while the words set is able to store only 5.3% of all the English words, the other data structures are able to store the entire dictionary, with a probability of false positive near to zero for bloom filters. Also, a secondary benefit is the access time required, in particular for bit string array and bloom

filters compared with the words set, in fact checking in the latter whether there is a word or not, requires to scan all the dictionary, while for the former ones require only to check that all the hashes' result are verified.

4.1 Optional point

As last point is useful to check the accuracy of the simulation regarding the optimal number of hashes compared to theoretical optimal number of hashes, computed as $k^{opt} = \frac{2^b}{w} \log 2$ where b is the number of bit used for the fingerprint and $w = 370103$, i.e. the length of the dictionary. The result is displayed in the chart below:

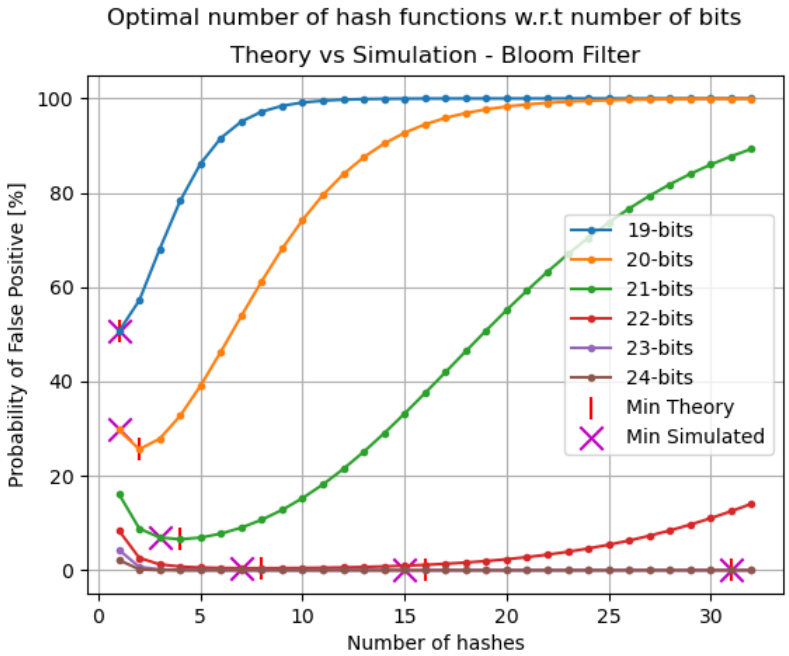


Figure 3: Comparison between simulated and theoretical optimal number of hash functions for bloom filters.

The simulation outputs and the theoretical results are very close, the difference between the two can be due to the necessary approximations in the computations.