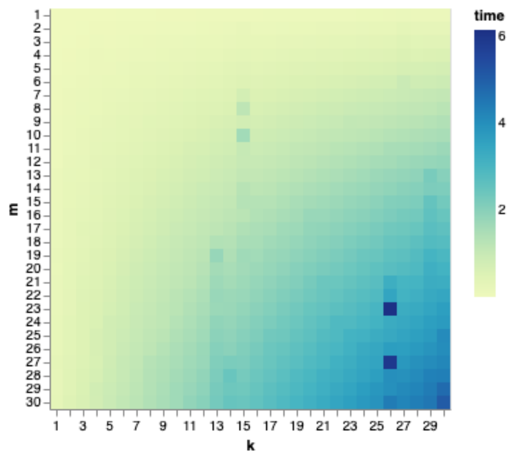
CAI Lab9

Jose Pérez Cano & Jordi Puig Rabat

November 2020

1 Task 1

1.1 Running time


(a) Reality

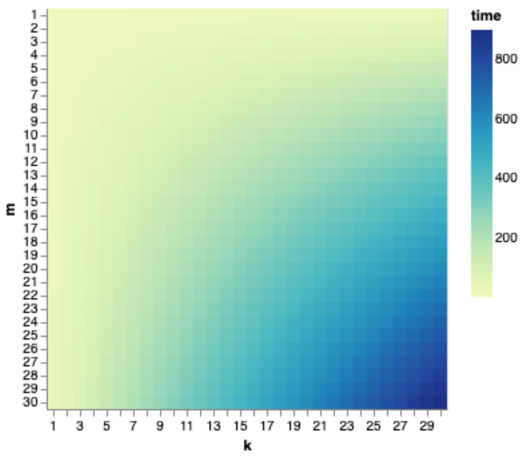

(b) Theory

Figure 1: Running time as a function of k and m

We can observe that the computed values are almost the same as expected except for some values that are a bit a higher. The reason for that is that we are selecting hashing functions at random, and it may happen that for a given execution the hashing lists aren't sufficiently separated and when joining them at the end we have a lot of repeated values or something similar. Also, any program is subject to the machine it is using and depending on the cache and other factors it could take more or less time.

1.2 Inference on k and m

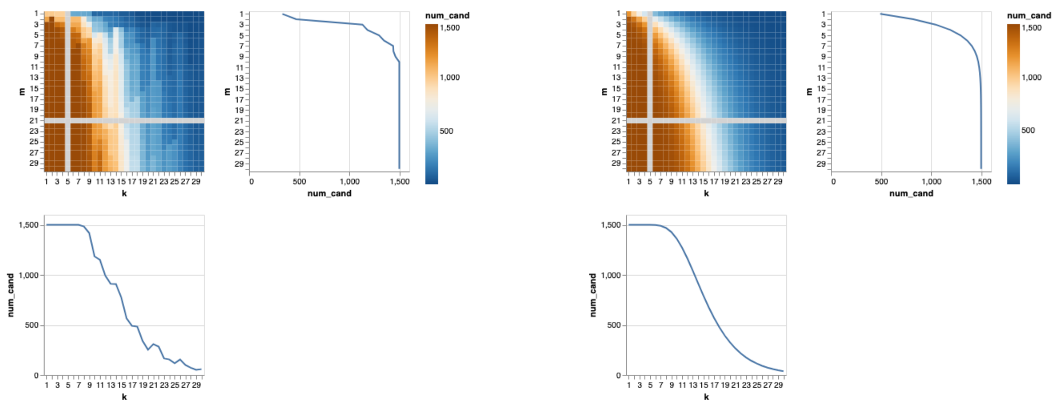
Trying some values we perceived that increasing k, the number of candidates decreases. A possible explanation is that joining several hashing functions has the result of needing more bits to be equal in order to be mapped together. This way, images will only be mapped together if the have more bits in common, which reduces the number of possible images.

Similarly, we observe an increase in candidates when m grows. The reason is that as more lists are appended, greater is the probability that one random image has the same hashing as the image we are considering. Therefore, repeating the hashing m times has as a consequence that more candidates are taken into account.

1.3 Size of candidate set

If we know the probability of collision of two images, let's call it P, then we could estimate the size of the candidate set as $f(k,m) = P \cdot N$ where $N = 1500$ is the total number of images. From the course we know that if s is the probability of two images having one bit identical (same value for some hashing function) then $P = 1 - (1 - s^k)^m$. Therefore we could estimate the size of the candidate set as

$$f(k,m) = (1 - (1 - s^k)^m) \cdot N$$



(a) Reality (b) Theory

Figure 2: Size of the candidate set as a function of k and m

Again, we present some graphics to compare theory and practice. This time the value of s need to be estimated somehow. A possible way could be

$$\hat{s} = \frac{1}{N^2} \sum_{i,j} s(d_i, d_j)$$

Here, we simply took a value that made the graphic more similar. Attached is the notebook which includes an interactive application to visualize the size of the function $f(k, m)$ when fixing k or m. At the top is an example.

2 Comparison between methods

Once we have programmed the different functions required we start to test and compare both methods. Our first approach was to use the given function `search(k, m, i)` and compare to the brute force function for each test image. The problem here was that the first function takes a lot of time because it hashes all the images every time. The correct solution is to create a single lsh(k, m) object at the beginning, iterate for all the test images comparing brute force to the method for LSH wich consist of finding the candidates and from there pick the best one. In some cases there was no candidates, this aspect must be taken into account when comparing the results also.

The next results were obtained with parameters of LSH k = 40 and m = 15. From the 296 test images 7 of them couldn't be matched with LSH; apart from that, the performance in time is way better in LSH than with brute force and the results are more or less similar. In our case the time spent with brute force is 5 times bigger than with the hashing method. The mean of the distances found is pretty similar: 79.82 with brute force and 86.91 with LSH; and the maximum of all the distances is 155 with brute force and 212 with LSH. Our conclusion of the comparison is that the imperfections LSH gives us (not always the best match) is worth the time it saves us. In this study the test set was small and the difference in time was by a factor of 5, but in terms of asymptotic complexity, brute force is quadratic while LSH is sublinear with the size of the data; and that is why we believe LSH is better than brute force.