LSH kNN graph for diffusion on image retrieval

Noname manuscript No.
(will be inserted by the editor)

LSH kNN Graph for Diffusion on Image Retrieval

Federico Magliani · Andrea Prati

the date of receipt and acceptance should be inserted later

Abstract Experimental results demonstrated the goodness of the diffusion mechanism for several computer vision tasks: image retrieval, semi-supervised and supervised learning, image classification. Diffusion requires the construction of a kNN graph in order to work. As predictable, the quality of the created graph influences the final results. Unfortunately, the larger the used dataset is, the more time the construction of the kNN graph takes, since the number of edges between nodes grows exponentially. A common and effective solution to deal with this problem is the brute-force method, but it requires a very long computation on large datasets. This paper proposes a technique, called LSH kNN graph, to efficiently create an approximate kNN graph which is demonstrated to be faster than other state-of-the-art methods (18x faster than brute force on a dataset of more than 100k images) for content-based image retrieval, while obtaining also comparable performance in terms of accuracy. LSH kNN graph has been tested and compared with the state-of-the-art approaches for image retrieval on several public datasets, such as Oxford5k, $\mathcal{R}$Oxford5k, Paris6k, $\mathcal{R}$Paris6k and Oxford105k.

Keywords Content-Based Image Retrieval · Diffusion · kNN graph

1 Introduction

Content-Based Image Retrieval (CBIR) is a research topic related to computer vision area. The problem focuses on the search for a query image in a dataset and rank the results based on the similarity to it. This image can be chosen or photographed by a mobile device. This problem seems simple to solve, but there are several challenges to be faced. The most significant ones are the robustness to orientation, scale and occlusion. With the recent advent of features extracted by means of Convolutional Neural Networks (CNN), it has been possible to obtain remarkable results, mitigating the effect of these problems. In parallel, several new embedding strategy were proposed

IMP lab - University of Parma, Parma, Italy E-mail: federico.magliani@studenti.unipr.it

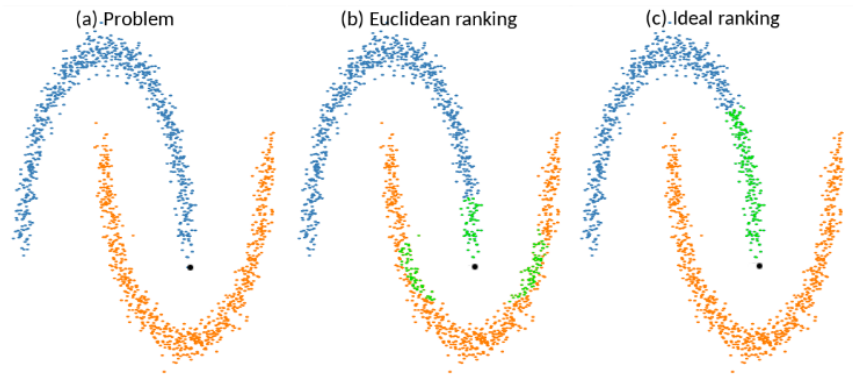

Fig. 1 In these three figures, two data distributions (orange and blue) are shown. In figure (a) the two distributions are depicted and the black point represents the query (which belongs to the blue class). In figure (b), the Euclidean distance is applied to rank the neighbours (green points) of the query, but several false positives are detected. On the other hand, in figure (c) the ideal ranking is shown. The diffusion can helps achieving this result, thanks to the propagation (diffusion) starting from the query image.

[26], [8], [19]. The combination of the new architecture for the feature extraction phase and the new method for the creation of global descriptors allowed to realise effective and efficient pipelines for CBIR problem, making feasible CBIR solutions also in case of large-scale datasets with reasonable retrieval time [18].

A recent breakthrough on this topic was made by graph theory, such as diffusion process, allowing to outperform the previous state of the art. In particular, the diffusion mechanism can be applied for retrieval task with stunning results [13], where the quality of the found query neighbours is better than checking the results in the Euclidean space. This process exploits the manifold distribution of the data through the creation of a graph, that represents the connection between dataset elements. The graph is mathematically represented by a pairwise affinity matrix [30]. Therefore, as also previously stated, the diffusion process requires the creation of a kNN graph of the embeddings used to represent the dataset images. Of course, the quality of the embeddings influences the results that can be achieved applying the diffusion process.

Once the kNN graph is created, the diffusion process works by finding (through random walks), for each node, the best path to reach the query, exploiting the weights of the traversed edges. The weights represent the similarity between the nodes connected by the edge (the greater the weight, the more similar the two nodes are).

Unfortunately, this approach also bears with it some drawbacks: (i) the setting of diffusion parameters is hard since they are dependent of the specific data distribution; and (ii) the time necessary to create the kNN graph can be unbearable for large datasets.

In practice, in order to apply diffusion a kNN graph needs to be created and its number of edges heavily influences the retrieval result. Moreover, it is hard to predict how much connected the graph needs to be to achieve good results. Therefore, the straightforward solution is to fully connect the nodes through the so-called brute-force strategy. This strategy is very easy to implement, but it tends to be very slow in

case of large datasets. Given a dataset of $\mathcal{N}$ images, the brute-force graph will have $\mathcal{N}^2$ edges, meaning that for $\mathcal{N} = 100k$, the number of edges of the brute-force graph will be equals to 10 billions.

As a consequence, different methods were proposed to solve this task in an approximate way, trying to create fastly an high quality approximate kNN graph [25], [5], [2], [29] (see Section 2 for further details).

Our proposed method called LSH kNN graph follows the principle that not all the connections between nodes in the graph are mandatory. It uses Locality Sensitive Hashing (LSH) projections [11] to subdivide the images contained in the dataset in many subsamples and then only the pair of images with a similarity greater than a threshold will be maintained and connected in the final graph. This process is repeated for each subsample and for different hash tables. The trade-off between the quality of the graph and the creation time is an important parameter of this method. In particular, the LSH kNN graph reaches the same or better retrieval results than many state-of-the-art algorithms on several public image datasets, but in much shorter time compared with the other methods.

The main contributions of this paper are:

- LSH kNN graph, that is an efficient algorithm for the creation of an approximate kNN graph. It can achieve the same or better performance, through the diffusion application, than several state-of-the-art algorithm in less time. Moreover a complexity analysis is presented in order to support the goodness of the proposed method.
- Several code optimizations for the creation of the graph are showed in order to reduce the computational time and the usage of memory.
- Experiments on several public image datasets and comparison with state-of-the-art methods.
- Improvements on the quality of the graphs due to some refinement techniques based on the neighbour propagation technique.

This paper is organised as follows. Section 2 introduces the general techniques used in the state of the art, while Section 3 reports some background information about ranking with diffusion. Next, Section 4 describes the proposed algorithm with a complete complexity analysis (Section 4.4) of the proposed method. Moreover, some refinement techniques (Section 4.5) are described and tested in order to demonstrate how it is possible to improve the quality of the graph with no extra effort. Then, Section 5 reports the experimental results on five public datasets: Oxford5k, $\mathcal{R}$Oxford5k, Paris6k, $\mathcal{R}$Paris6k and Oxford105k. Finally, concluding remarks are reported.

2 Related work

Recently, several graph applications in computer vision tasks have been proposed in the literature: diffusion for retrieval [13], unsupervised or semi-supervised training [12, 6], image classification [15] and manifold embedding [28].

Similarly to our case, all these papers use the k-Nearest Neighbour (kNN) to create the graph. More formally, you can describe the undirected graph G with $G(V, E)$,

where V represents the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and E represents the set of edges $E = \{e_1, e_2, \ldots, e_n\}$. The nodes represent all the images in the dataset and the edges represent the connections between nodes. The weight of each edge determines how much the two images are similar: the higher the weight, the more similar the two images are. The weights of the edges are set with the cosine similarity calculated between the embeddings.

The problem of creating the kNN graph differs from the nearest neighbours search task since it does not need to index all the dataset image in order to fastly retrieve the elements similar to the query image, but it needs to create the relations between all the similar images in the dataset [2]. For example, PQ [14] and BoI [17] are nearest neighbours methods, but they are not suitable for this task due to data structure adopted, not graph based. After the creation of the graph, the application of some heuristic allows to extrapolate useful information through the graph for improving the performance of the retrieval system or the image classifier.

Different solutions are available in the literature to efficiently create the kNN graph. The most simple is the exact or brute-force method. The advantages of this methods are that is simple to implement and that obtains usually the best results. Unfortunately, it requires very long time to compute.

Alternatively, approximate kNN graph algorithms want to speed up the process, but maintaining good performance after diffusion application. They can be subdivided in two families of strategies: algorithms based on divide and conquer strategy and techniques based on local search optimizations (e.g., NN-descent [5]). As the name says, divide and conquer is composed by two steps: firstly, based on a certain heuristic, the images in the dataset are divided in subsamples and then for each subsample a kNN graph is created. In the end, all the created subgraphs are merged, obtaining the final kNN graph. Naturally, the number of subdivisions influences the final performance and the computational time of the approximate kNN graph algorithm. Moreover, the heuristic used for the subdivision task is crucial for the method and needs to be very effective and efficient. For instance, the well-known K-means algorithm [1], while being widely and successfully used for clustering, is too slow for this task. To solve this problem, the method proposed in this paper is faster than K-means.

An interesting work [29] following the divide and conquer strategy exploits LSH (Locality-sensitive hashing) to create the approximate kNN graph by using spectral decomposition of a low-rank graph matrix. Instead, Chen *et al.*[2] follow the same strategy, but applying recursive Lanczos bisection. In this case, two divide steps are proposed: the overlap and the glue method. The difference between the two proposed techniques is on the subsets, overlapped for the former and disjointed for the latter. Another interesting paper from Wang *et al.*[27] proposes an algorithm for the creation of an approximate kNN graph based on random collections of dataset elements. Repeating many times this process allows to theoretically cover the entire dataset.

On the other hand, the methods based on local optimizations are based on the principle that "a neighbour of my neighbour is my neighbour", introduced by Dong *et al.*with NN-descent [5]. Starting from a random Nearest Neighbour (NN) list for each node, the method iteratively tries to update these lists. The update process is very simple: for a node a, the algorithm finds two neighbours (b and c) and then

tries to update the NN list of b with the distance $d(a,b)$ and the NN list of c with the distance $d(a,c)$. The process is repeated until the number of updates executed on the NN lists is less than a threshold, selected as parameter of the algorithm. A weakness of this method is the correct setting of the initial dimension of the NN lists and the number of updates to execute on them. In fact, if the dimension of the lists is large or the number of updates is very high, the method will require very long time to compute the kNN graph. Different works tried to adapt the NN-descent to their specific application domains [21], [9], [4].

Finally, a mixed solution, called *Random Pair Division*, based on both divide-and-conquer strategy and NN-descent was proposed by Sieranoja *et al*.[25]. The first step is the subdivision of the dataset elements in order to speedup the subgraphs creation. The heuristic adopted is very simple: starting from two random dataset elements, all the elements will be assigned to one of the two sets based on the distance to the initial random selected element. The process is repeated if the size of one set is greater than a threshold. After that, the subgraphs are created on the elements contained in the subsamples using the brute-force approach. In addition, the NN-descent is applied to improve the quality of the graph and to connect also elements of different subgraphs.

3 Ranking with diffusion

The diffusion is a mechanism that allows to spread the query similarities on a graph, created with the dataset elements, in order to catch all the neighbours elements to the query. The diffusion application requires the presence of the affinity matrix.

The affinity matrix A is the adjacency matrix of a weighted undirected graph G. It is symmetric ($A = A^T$), positive ($A > 0$) and with zero self-similarities ($diag(A) = 0$). In order to apply diffusion, it is worth to calculate the Laplacian of the graph $\mathcal{L} = D - A$, where $D = diag(A1_n)$ is the degree of the graph and $A1_n$ is the diagonal matrix with the row-wise sum of A. Then, it is common to normalize the affinity matrix for obtaining the transition matrix $S = D^{-1/2}AD^{-1/2}$ and the Laplacian $\mathcal{L} = I_n - S$ where I_n indicates the identity matrix, that has size equals to n.

After the creation of the Laplacian and the relative normalization, Zhou *et al*.[30] proposed to apply diffusion for retrieval purposes starting from the query points. They created a vector $y = (y_i) \in \mathbb{R}^n$ in this way:

$$y_i = \begin{cases} 1 & if \quad x_i \quad is \quad a \quad query \\ 0 & otherwise \end{cases}$$

The objective of ranking with diffusion is to find the neighbours of a query, therefore a ranking function $f = (f_i) \in \mathbb{R}^n$, that allows to generate a vector with the similarity score of each image x_i to the query, is created. It is worth to note that this process needs to be repeated for each query. The diffusion mechanism can be represented in the following way by the ranking function:

$$f^t = \alpha S f^{t-1} + (1 - \alpha)y$$

The ranking function defines the random walk process on the graph, while α indicates the probability to jump on an adjacent vertex according to the distribution

S and $(1 - \alpha)$ indicates the probability to jump to a query point. At the beginning of this process the ranking function is initialised with the value obtained from the application of the Euclidean distance. Repeating many times this process allows for each point to spread their ranking score to their neighbours in the graph. Exploiting this principle it is possible to better capture the manifold structure of the dataset than applying the Euclidean distance.

4 Proposed approach

LSH kNN graph adopts LSH to subdivide in subsets the global descriptors representing the images of the dataset. The number of the subsets depends to the hash dimension used for the projection phase and the size of each set usually depends to the dataset size because the subdivision is pretty much similar in each bucket. In the following, first the hashing technique is introduced and then the entire algorithm is described.

4.1 Notations and background of LSH

Locality-Sensitive Hashing (LSH) [11] is an hashing technique based on the principle that similar points will be close also in the projected space with high probability.

The LSH function for Hamming space is a scalar projection:

$$h(\mathbf{x_f}) = sign(\mathbf{x_f} \cdot \mathbf{p}) \tag{1}$$

where $\mathbf{x_f}$ is the feature vector and $\mathbf{p}$ is a vector with the components randomly selected from a Gaussian distribution $\mathcal{N}(0, 1)$, called *projection function*.

This process can be repeated many times (L represents the number of hash tables used in the LSH process) in order to improve the quality of the projections, using different Gaussian distribution.

A common LSH application for retrieval purposes can be summarised with these three steps:

1. project all the database descriptors using different Gaussian distributions;
2. for each query, project the image descriptor using the same Gaussian distributions adopted for the database elements;
3. search and rank in the hash table buckets the database images.

Many other hashing techniques have been proposed and implemented. For example, the multi-probe LSH [16] tries to reduce the number of hash tables used for the projections, exploiting the fundamental principle of LSH that similar items will be projected in the same buckets or in near buckets with high probability. This idea is implemented checking, during the search phase, also the buckets near the query bucket. Sadly, the performance improvement determines, as a consequence, an increase of the computational time.

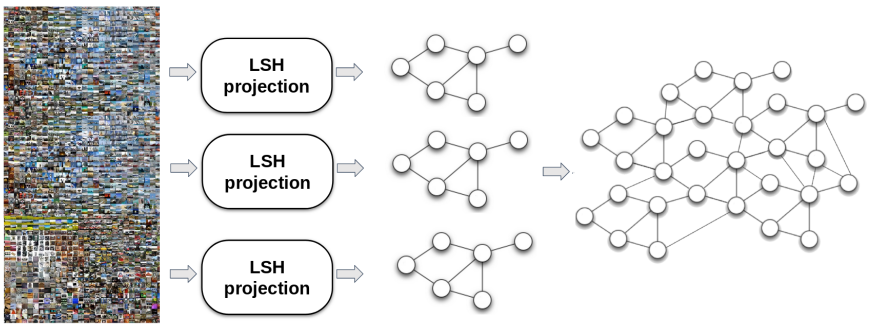

Fig. 2 Pipeline of a basic algorithm for kNN graph construction based on LSH projections and following the divide-and-conquer strategy.

4.2 LSH kNN graph

LSH kNN graph creates an undirected kNN graph G from a dataset $\mathcal{S} = \{s_1, \ldots, s_\mathcal{N}\}$ of $\mathcal{N}$ images. To create the graph and connect the nodes through edges, a similarity measure $\theta : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ is adopted. The connection between the nodes i and j in the graph is calculated with the similarity measure $\theta(s_i, s_j) = \theta(s_j, s_i)$. There are different techniques to calculate the similarity measure. For our purpose we adopted the cosine similarity, that can be calculated with the dot product between the global image descriptors of the dataset images. The proposed approach follows the divide-and-conquer strategy since the first step is the split of the dataset elements in many subsets based on LSH projections, as showed in Fig. 2. As previously reported, LSH allows to project similar elements in the same bucket in a projected space. Exploiting this principle it is possible to created a set of buckets $B = \{B_1, \ldots, B_m\}$ from several hash tables. In addition, the use of more or less bits (δ) for the projection step influences the quality of the results and the final number of the buckets. Considering also the number of the hash tables (L) adopted for the projection, the total number of buckets will be $N = 2^\delta \cdot L = |B| \cdot L$. We will indicate the n elements of the i-th bucket B_i as follows: $B_i = \{b_{i1}, \ldots, b_{in}\}$. There are no guarantees that all the similar elements will be in the same bucket because this approach represents an approximate solution. As a consequence, a good idea is to try to find a trade-off between the number of the buckets for each hash table (2^δ), by tuning the bits used (δ) for the projection step and the number of hash tables (L). Usually, if the objective is to project more elements in the same bucket, a good solution is to use a small number of buckets. It allows to reduce the time spent in the divide phase, but, on the other hand, the conquer phase will require more time to be executed. On the other hand, with more bits adopted for the projections, and thus more buckets for hash tables, the divide step will be lightly slower, but the conquer one will be faster.

The conquer step provides the connection between the elements in each bucket. During this phase, the pipeline connects the dataset elements and stores in memory the final graph. In this case, the method adopted to solve this subtask is the brute-force approach, so all the elements in the bucket are connected, creating a kNN graph $G = (V, E)$, where $V = (b_{x1}, \ldots, b_{xn})$ where $x = 1, \ldots, m$ and E is the set of edges

with weights computed with the similarity θ: $E = \{\forall\,(b_{xi}, b_{xj}) \in B_i : \theta\,(b_{xi}, b_{xj})\}$ where $x = 1, \ldots, m$. The key point here is that applying brute-force several times but on smaller sets results at the end to be faster than applying it once but on the entire, larger set of data. Morever, differently from other methods based on the divide-and-conquer strategy, no final merge between all the subgraphs is required, since in our case a single graph is created and updated with new connections. For more details on the implementation, please check Appendix A.

4.3 Multi-probe LSH kNN graph

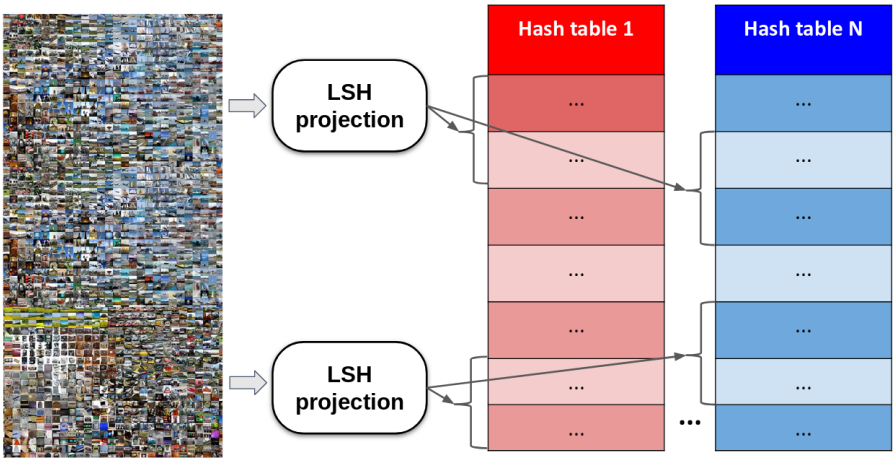

Fig. 3 Pipeline of multi-probe LSH kNN graph. Best viewed in colour.

In addition to the basic LSH kNN graph described in the previous section, a multi-probe version of it is also here proposed. This method exploits the principle of multi-probe LSH with the objective of reducing the number of hash tables used.

Multi-probe LSH [16], during the query phase, checks also buckets near the query bucket b_{query} because they probably contain similar elements to the ones contained in it. For our purpose, this idea can be exploited during the projection step. It means that each dataset elements, after the hashing phase, it will be projected also in the neighbours buckets, as showed in Fig. 3. The process will be lightly slower due to the greater number of projections to be performed. In order to maintain a good trade-off between quality of the graph and computational time, the elements will be projected only in the 1-neighbourhood. It is worth to note that the buckets are constructed using binary numbers, so the Hamming distance can be exploited. As a consequence, 1-neighbourhood represents the set of buckets with Hamming distance less or equal to 1 ($H_d(b_{xi}, b_{xj}) \leq 1$).

More formally, the elements obtained with the application of the multi-probe LSH are the followings:

$$B_{multi-probe} = \{b_{x1}, \ldots, b_{xn}\} : \\ H_d(b_{query}, b_{xj}) \leq 1 \wedge b_{xj} \in B; x = 1, \ldots, m, j = 1, \ldots, n$$

Similarly to the basic LSK kNN graph, the growth of the bits used for the hashing task directly influences the number of neighbours available in each bucket in this way: $\sum_{i=0}^{n} \binom{\log_2 \delta}{i}$.

Although usually the final results are better than the ones obtained by the previous method, the total computational time needed by this approach is greater as well. A possible solution for this problem is represented by using a percentage γ that allows to unsupervisely decide to project or not the elements also in the 1-neighbourhood. For example, by setting $\gamma = 50\%$, only half of the elements will be projected also in the 1-neighbourhood buckets. Empirically, it has been found that the best trade-off is reached using $\gamma = 50\%$, which will be the value used in in all our experiments.

4.4 Complexity analysis

In this section, we will briefly analyse the complexity of the proposed methods.

For the projections phase when LSH is applied, the complexity will be $O(\delta \cdot \Delta \cdot L \cdot \mathcal{N})$, where $\mathcal{N}$ is the number of images in the dataset, δ is the number of bits used in each projection, L is the number of hash tables and Δ represents the dimension of the embedding used for the representation of the input image. In the case of multi-probe LSH, the complexity will be greater because each image is projected in more buckets: $O(\delta \cdot \Delta \cdot L \cdot (\mathcal{N} \cdot \gamma \cdot L) \cdot \mathcal{N})$.

Then, the calculation of the similarity measure of all the possible pairs of elements contained in a bucket has a complexity of $O(n^2 \cdot 2^{\delta} \cdot L)$, where n represents the number of elements found in the bucket. By hypothizing a uniform distribution of buckets, the value of n can be approximated as: $n \sim \frac{\mathcal{N}}{2^{\delta}}$.

For supporting this hypothesis, Figures 4 and 5 show the LSH distributions (for different values of δ) on Oxford5k dataset (see Section 5). The values reported in each graph represent the distribution of the database elements in the buckets. The final obtained distribution resembles a sort of Gaussian.

The complexity needed for the combination of the subgraphs (conquer phase) is negligible because all the subgrahps are directly appended on the final graph.

To conclude, the final complexity of the proposed approaches can be obtained by summing the single components:

- for basic LSH kNN graph approach: $O(\delta \cdot \Delta \cdot L \cdot \mathcal{N}) + O(n^2 \cdot 2^{\delta} \cdot L)$ which can be further simplified (exploiting the approximation of n mentioned before) in $O(\frac{L \cdot \mathcal{N}^2}{4} + L \cdot \mathcal{N} \cdot \delta \cdot \Delta)$;
- for multi-probe LSH kNN graph approach: $O\left(\delta \cdot \Delta \cdot L \cdot (\mathcal{N} \cdot \gamma \cdot L) \cdot \mathcal{N}\right) + O(n^2 \cdot 2^{\delta} \cdot L)$ which can be further simplified as before and also removing lower order terms in $O(L^2 \cdot \mathcal{N}^2 \cdot \delta \cdot \Delta \cdot \gamma)$.

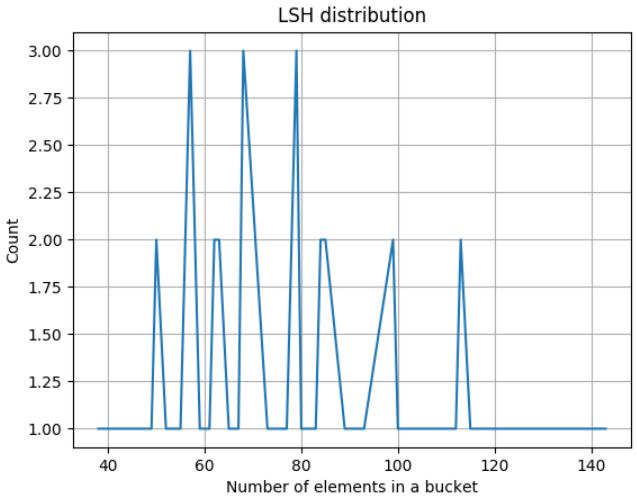

Fig. 4 Distribution of dataset images projected through LSH on Oxford5k with $\delta = 6$ and $L = 20$.

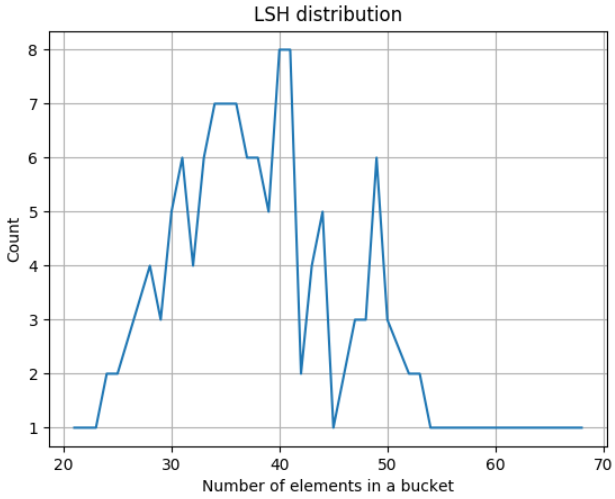

Fig. 5 Distribution of dataset images projected through LSH on Oxford5k with $\delta = 7$ and $L = 20$.

Therefore, it is evident that while basic LSH kNN approach is bounded $O\left(L \cdot \mathcal{N}^2\right)$, multi-probe version is, as expected, more computationally complex and bounded $O\left(L^2 \cdot \mathcal{N}^2\right)$.

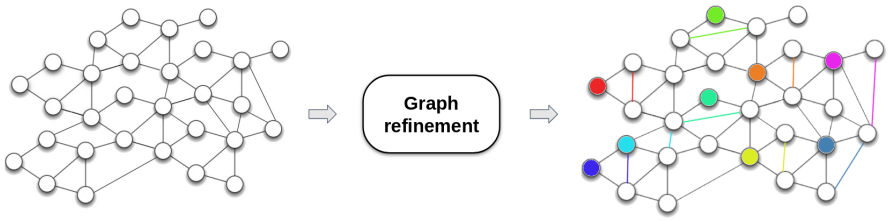

Fig. 6 Example of the working of the graph refinement techniques. Some extra edges are added in order to improve the quality of the final kNN graph. The new connections are coloured with the same colour of the relative nodes in order to make clear the working idea of the algorithm. Best viewed in colour.

4.5 Graph refinement

Graph refinement or neighbour propagation is an important step during the kNN graph creation task. It allows to refine the quality of the graph in order to improve the final results. In general, the algorithm aims at adding more edges between nodes in the graph (as shown in the Fig. 6) since, hopefully, these edges will be improve the diffusion result. Unfortunately, these improvements require an extra effort and the final computation time will be greater.

The most diffused graph refinement method is one-step neighbour propagation [5]. It is an iterative process, in which the neighbours of neighbours are checked. In other words, if a is a neighbour of b and b is a neighbours of c, then is likely that a is a neighbour of c. This approach requires the maintenance of a kNN list of each node. For each node, two neighbours are randomly picked and then connected if their similarity is greater than the worst in the list, by also updating the other kNN lists accordingly. This process continue until the number of updates on the kNN lists surpasses a threshold value.

In this paper we propose a novel method called *sorted neighbour propagation*, that represents an improvement to the previously presented technique. The kNN lists are sorted based on the similarity obtained during the creation of the kNN graph and then only the $topN$ elements are evaluated. All the possible pairs of neighbours found in these $topN$ elements are added to the graph. Experiments in the next section will show the performance of the proposed method on different public image datasets compared to other state-of-the-art techniques, such as: kNN graph without graph refinement (as a baseline), random propagation and one-step neighbour propagation.

5 Experimental results

Previous works have evaluated the methods for creating approximate kNN graphs by checking the number of common edges between the approximate and the exact kNN graph. Instead, our aim is to evaluate the complete kNN graph pipelines after the diffusion and retrieval modules. The rationale of this choice lies in our objective to evaluate how effective (and efficient) are our proposals for the approximate kNN graph creation in terms of retrieval accuracy when diffusion is applied.

The features used in all the experiments for the creation of the kNN graphs are R-MAC descriptors [13].

The hardware adopted for the experiments is the following: CPU Intel Core i7 @ 3.40 GHz x 8, 32Gb RAM DDR4.

5.1 Datasets

There are many different image datasets for Content-Based Image Retrieval that are used in order to evaluate the algorithms. The most used are the following:

- **Oxford5k** [22] is composed by 5063 images representing the buildings and the places of Oxford (UK), subdivided in 11 classes. All the images are used as database images and the query images are 55, which are cropped for making more difficult the querying phase;
- $\mathcal{R}$**Oxford5k** [24] is composed by 4993 images. This dataset represents the revisited version of the previous one. It is composed by 70 queries, that are new images added to the old dataset. All the images are labelled in order to test the pipeline at 3 different retrieval difficulties: *Easy*, *Medium* and *Hard*;
- **Paris6k** [23] is composed by 6412 images representing the buildings and the places of Paris (France), subdivided in 12 classes. All the images are used as database images and the query images are 55, which are cropped for making more difficult the querying phase;
- $\mathcal{R}$**Paris6k** [24] is composed by 6322 images. As before, this dataset represents the revisited version of the previous one, with 70 additional queries and the same three difficulties: *Easy*, *Medium* and *Hard*;
- **Flickr1M** [10] contains 1 million Flickr images under the Creative Commons license. It is used for large scale evaluation. The images are divided in multiple classes and are not specifically selected for the image retrieval task.

Moreover, with the addition of 100k images of Flickr1M it is possible to create **Oxford105k** datasets.

5.2 Evaluation metrics

To evaluate the accuracy in the retrieval phase, mean Average Precision (mAP) is used on all the image datasets used. The mAP is the mean of average precision that identifies how many elements that finds are relevant to the query image. In order to compare a query image with the database, L_2 distance is employed.

5.3 The importance of diffusion for retrieval

Before starting the evaluation of the approximate kNN graph pipelines, it is worth to better motivate our choice of using diffusion on graphs and, therefore, the need for an efficient creation of an approximate kNN graph.

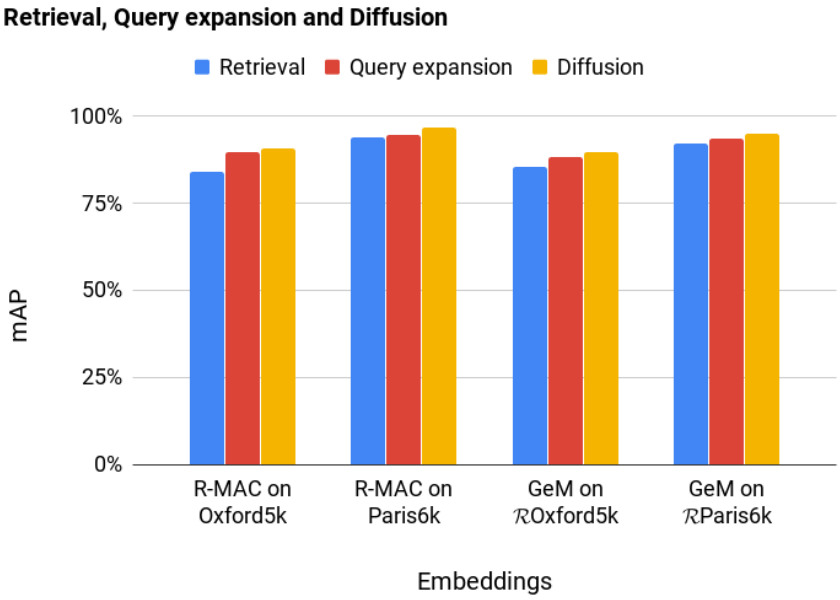

Fig. 7 Comparison of results obtained using R-MAC descriptors [13] tested with different approaches on Oxford5k, Paris6k, $\mathcal{R}$Oxford5k and $\mathcal{R}$Paris6k.

Fig. 7 shows some experiments performed on both Oxford5k, Paris6k, $\mathcal{R}$Oxford5k and $\mathcal{R}$Paris6k datasets. In each experiment the query expansion is executed using the top 5 elements of the query ranking, following the original approach [3]. The diffusion parameters for Oxford5k and Paris6k (using global descriptors) are set after an optimization process based on genetic algorithms [20] in this way:

- $\alpha = 0.97$. It indicates the contributions to the ranking score from the neighbours.
- $\beta = 3$. It indicates the exponentiation of the affinity matrix elements.
- $\gamma = 1$. It indicates the exponentiation of the query vector elements.
- $iterations = 10$. It represents the maximum number of iterations necessary for the resolution of the equation $A * f = y$ in the diffusion process through the application of the conjugate gradient. A represents the affinity matrix of the dataset elements, instead y identifies the query vector and f is the ranking vector.
- $k_s = 97$. It represents the maximum number of node to cross during the random walk process.
- $k = 7$. It represents the number of neighbours to find.
- $truncation = 4000$. It represents the best number of rows to use for diffusion.

Fig. 7 demonstrates clearly that diffusion can bring significant improvements in the retrieval accuracy.

5.4 Results on Oxford5k

Method	LSH projection	Graph creation	mAP
LSH kNN graph	**0.45 s**	**0.52 s**	**94.45%**
multi-probe LSH kNN graph	0.29 s	1.54 s	94.13%
NN-descent [5]*	-	55 s	85.81%
RP-div [25]*	-	1.16 s	84.68%
Wang *et al*.[27]*	-	1.5 s	92.60%
brute-force	-	2.01 s	92.79%

Table 1 Comparison of different approaches of kNN graph creation tested on Oxford5k using different type of embeddings. * indicates that the method is a C++ re-implementation.

Table 1 reports the retrieval results obtained with the diffusion on different kNN graphs, constructed adopting several algorithms. As a result, the different values for the LSH parameters (δ and L) produces different final retrieval results. For LSH kNN graph approach, the best combination is $\delta = 6$ and $L = 20$, instead for multi-probe LSH kNN graph method is $\delta = 6$ and $L = 2$. It is evident that LSH kNN graph approach obtains the best trade-off between performance and computational time needed (also considering the sum of LSH projection and graph creation) for the entire process.

Conversely, NN-descent [5] achieves poor results. In this case, the number of neighbours evaluated in the NN-descent process is set to 75, meaning that each kNN list is composed by 75 elements. In addition, this approach resulted to be the slowest one with 55 secs needed to create the kNN graph.

Also RP-div [25] obtains quite poor results, but in a fast way. This is probably due to the nature of this method, based on the randomness of the points used for the divide step, which speeds up the graph creation process, but does not allow to have good retrieval accuracy. The maximum size of each set is 50, meaning that every set larger than 50 elements is further split.

The last compared approach is the method proposed by Wang *et al*.[27], whose computational time is not very high, probably because the elements are randomly chosen. It also achieves good accuracy, although the final quality of the graph depends too much to the random choice of the elements. In this case, the number of iterations are set to 1000 and the elements for each set are 200.

Lastly, the basic brute-force method resulted to have a good trade-off between accuracy and efficiency.

Table 2 reports the results obtained after the application of some graph refinement techniques adopted on the baseline LSH kNN graph. First row shows the baseline method with no graph refinement. Random propagation (second and third rows) randomly adds some new edges in the graph (100 and 500 new edges, respectively). This graph refinement techniques improves significantly the mAP with a limited increase in time needed for graph creation. Similar considerations can be done for one-step neighbour propagation, which requires some more extra time to create the graph, while obtaining results comparable with random propagation. Our proposed method

Method	TopN	Graph creation	mAP
LSH kNN graph	-	0.11 s	78.34%
Random propagation	100	0.33 s	85.77%
Random propagation	500	1.10 s	90.73%
One-step neighbour propagation [5]	100	0.45 s	87.23%
One-step neighbour propagation [5]	500	1.78 s	89.84%
Sorted neighbour propagation	100	10.66 s	**91.66%**

Table 2 Comparison of different techniques for graph refinement adopted on the baseline LSH kNN graph for Oxford5k.

(last row) obtains the best precision in retrieval, but at a much higher cost in terms of computational time, that is related to the sort operation executed in order to connect only interesting and useful nodes in the kNN graph. Please note that in this case we used $\delta = 6$ and $L = 2$ (instead of $\delta = 6$ and $L = 20$ of the previous Table), resulting in different graph creation time for our approach.

5.5 Results on $\mathcal{R}$Oxford5k

Method	LSH proj.	Graph creation	Easy	Medium	Hard
LSH kNN graph	**0.48 s**	**0.52 s**	**90.51%**	**76.64%**	**54.26%**
multi-probe LSH kNN graph	0.27 s	0.73 s	90.62%	74.96%	49.77%
NN-descent [5] *	-	86 s	78.08%	65.36%	39.62%
RP-div [25] *	-	1.13 s	85.03%	70.66%	45.79%
Wang *et al*.[27]*	-	1.13 s	89.04%	74.28%	50.15%
brute-force	-	1.3 s	89.51%	76.55%	53.10%

Table 3 Comparison of different approaches of kNN graph creation tested on $\mathcal{R}$Oxford5k using different type of embeddings. * indicates that the method is a C++ re-implementation.

We made the same experiments also for $\mathcal{R}$Oxford5k dataset. Table 3 reports the retrieval results and the computational time. The parameters of all the approaches are kept unchanged wrt the previous dataset, except for Wang *et al*.[27], in which the number of iterations is increased to 400 and the elements of each set are 200. As mentioned before, in this case three different mAP values are shown accounting for the three different difficulties.

The proposed approaches (first two rows) exhibit the best trade-off between accuracy and efficiency for the all the three cases, with comparable (even slightly better) results wrt brute-force. The other approaches confirm they generally-poor performance when both the measures (time and accuracy) are considered.

Table 4 reports the results for graph refinement techniques. The quality of the graph obtained with the proposed sorted neighbour propagation method outperforms the other methods in all the three cases, but still suffers from a higher computational time. It is still worth remembering (as for the previous dataset) that this experiment has been conducted with $\delta = 6$ and $L = 2$.

Method	TopN	Graph creation	Easy	Medium	Hard
LSH kNN graph	-	0.09 s	80.46%	64.94%	39.50%
Random propagation	100	0.33 s	83.75%	68.42%	39.39%
Random propagation	500	1.06 s	88.43%	73.33%	47.50%
One-step neighbour propagation [5]	100	0.42 s	83.73%	68.32%	41.47%
One-step neighbour propagation [5]	500	1.71 s	89.28%	74.08%	50.07%
Sorted neighbour propagation	100	10.72 s	**89.68%**	**74.95%**	**50.50%**

Table 4 Comparison of different techniques for graph refinement adopted on the baseline LSH kNN graph for $\mathcal{R}$Oxford5k.

5.6 Results on Paris6k

Method	LSH projection	Graph creation	mAP
LSH kNN graph	**1 s**	**0.80 s**	**97.32%**
multi LSH kNN graph	0.35 s	2.28 s	97.01%
NN-descent [5] *	-	60.10 s	94.44%
RP-div [25] *	-	3.63 s	96.65%
Wang *et al*.[27]*	-	1.95 s	96.85%
brute-force	-	2.61 s	96.93%

Table 5 Comparison of different approaches of kNN graph creation tested on Paris6k. * indicates that the method is a C++ re-implementation.

Similar results and conclusions are obtained for Paris6k dataset. In fact, Table 5 shows that the proposed methods (especially basic LSH kNN graph) achieved the best results in terms of mAP in less time compared to the time needed by the brute-force approach. The configurations of each algorithm are the same of the previous datasets. Moreover, Table 6, comparing graph refinement techniques, also leads to similar considerations as before, where the proposed method (last row) gets the best accuracy in all the cases, but with higher computational cost.

Method	TopN	Graph creation	mAP
LSH kNN graph	-	0.2 s	92.80%
Random propagation	100	0.45 s	83.85%
Random propagation	500	1.42 s	94.70%
One-step neighbour propagation [5]	100	0.66 s	85.01%
One-step neighbour propagation [5]	500	2.43 s	94.70%
Sorted neighbour propagation	100	13.38 s	**97.27%**

Table 6 Comparison of different techniques for graph refinement adopted on the baseline LSH kNN graph for Paris6k.

5.7 Results on $\mathcal{R}$Paris6k

Method	LSH proj.	Graph creation	Easy	Medium	Hard
LSH kNN graph	0.60 s	0.66 s	**95.09%**	**92.00%**	**83.05%**
multi LSH kNN graph	0.38 s	0.92 s	94.46%	88.49%	74.08%
NN-descent [5] *	-	104 s	94.16%	89.50%	80.78%
RP-div [25] *	-	1.23 s	77.02%	67.80%	55.70%
Wang *et al.*[27]*	-	1.43 s	93.09%	85.48%	70.15%
brute-force	-	1.56 s	94.86%	91.13%	82.78%

Table 7 Comparison of different approaches of kNN graph creation tested on $\mathcal{R}$Paris6k using different type of embeddings. * indicates that the method is a C++ re-implementation.

Table 7 presents the results obtained and the computational time on $\mathcal{R}$Paris6k dataset. The performance are similar to the ones obtained on $\mathcal{R}$Oxford5k, with our approach outperforming brute force. The proposed approach is slightly faster than brute-force.

Same conclusions of the previous datasets can be drawn for graph refinement techniques for this dataset (Table 8).

Method	TopN	Graph creation	Easy	Medium	Hard
LSH kNN graph	-	0.12 s	89.24%	81.30%	63.16%
Random propagation	100	0.37 s	92.91%	86.93%	71.94%
Random propagation	500	1.41 s	94.36%	89.26%	75.52%
One-step neighbour propagation [5]	100	0.56 s	92.70%	86.79%	71.69%
One-step neighbour propagation [5]	500	2.24 s	94.77%	89.30%	75.85%
Sorted neighbour propagation	100	10.72 s	**94.90%**	**89.70%**	**76.60%**

Table 8 Comparison of different techniques for graph refinement adopted on the baseline LSH kNN graph for $\mathcal{R}$Paris6k.

5.8 Results on Oxford105k

Method	LSH projection	Graph creation	mAP
LSH kNN graph	**23 s**	**77 s**	**94.20%**
multi-probe LSH kNN graph	5 s	420 s	93.85%
Wang *et al.*[27]*	-	150 s	91.70%
brute-force	-	10560 s	93.05%

Table 9 Comparison of different approaches of kNN graph creation tested on Oxford105k. * indicates that the method is a C++ re-implementation.

Finally, this section reports the results on a larger dataset, Oxford105k. Unfortunately, in this case graph refinement techniques can not be tested due to our limited hardware resources. Moreover, we have conducted tests for RP-div [25] and NN-descent [5] since they already demonstrated their limited performance on smaller datasets.

Therefore, Table 9 presents the result of the experiments executed on Oxford105k. The growth of the dataset size influences the graph creation time, but it is worth noting how our approaches scale better than brute-force, by keeping the total computational time at 100 seconds and still achieving better accuracy than brute force.

6 Conclusions

In this paper we presented an algorithm called LSH kNN graph for the creation of an approximate kNN graph exploiting LSH projections. The proposed method follows the divide-and-conquer strategy: the dataset elements are subdivided through the use of an unsupervised hashing function and then in each subset a subgraph is created using the brute-force approach. The proposed approach obtains the same or better results than other state-of-the-art methods, but in less time. Regarding the memory footprint, the implementation with sparse matrices combined with other code implementations (see Appendix A) have allowed to achieve very good results with limited memory requirements.

Moreover, multi-probe LSH kNN graph algorithm is proposed based on the principle of multi-probe LSH. In this case, the elements are projected also in the 1-neighbourhood buckets, allowing to reduce the number of hash tables needed, while almost preserving the overall accuracy.

To support the goodness of the proposed algorithms, a complexity analysis is also presented. Finally, a new graph refinement technique is introduced in order to boost the quality of the final graph with the addition to the graph of new useful connections between nodes. Compared with other graph refinemtn techniques, the proposed sorted neighbour propagation achieves the best result, but with an extra time effort.

As a future work, we are implementing a distribute version of these approaches with the objective of executing them on large-scale datasets.

Acknowledgment

This is work is partially funded by Regione Emilia Romagna under the “Piano triennale alte competenze per la ricerca, il trasferimento tecnologico e l’imprenditorialità”.

Appendix A Implementation details of LSH kNN graph approach

The proposed method uses LSH projections for the creation of the approximate kNN graph. The main advantages of LSH are the simplicity to use and the speed of the method. For example, apply LSH on 100k images in C++ needs only 10 seconds. It is worth to note that the variation of the values of the LSH parameters can change considerably the final performance. For both the two parameters (δ and L), in order to find the best combination, it is suggestable to execute several experiments.

The projection algorithm works as follows. For each bit we executed the dot product between the image descriptor and the corresponding projection vector. If the results is positive, the value of the projected bucket is increased by a power of two. For example, considering a hash table composed by 8 buckets ($\delta = 3$) and the first dot product negative, the second and the third positive, the element will be projected in the sixth bucket, because $6 = (2^0) \cdot \mathbf{0} + (2^1) \cdot \mathbf{1} + (2^2) \cdot \mathbf{1}$. This process will be executed for each hash table and for all the image descriptors.

Two implementation variants of our LSH kNN graph are proposed. From now, the kNN graph will be represented by the affinity matrix A, that represents the weight edges between all the nodes. This abstraction can help for the implementation of the algorithms.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{NN} \end{bmatrix}$$

Furthermore, not all the similarities are useful for the diffusion process, suggesting to remove or avoid to insert edges with weight less than a threshold (th), without jeopardising the final retrieval performance. From our experiments, this threshold can be set to 0.3.

procedure LSH KNN GRAPH
  $th \leftarrow 0.3$
  for $a_{ij} \in A$ **do**
    $a_{ij} \leftarrow 0.0$
  end
  for $B_x \in B$ **do**
    for $b_{ix} \in B_x$ **do**
      for $b_{iy} \in B_x$ **do**
        if $\theta(b_{ix}, b_{iy}) \geq th$ **then**
          $a_{ij} \leftarrow \theta(b_{ix}, b_{iy})$
        end
      end
    end
  end
end procedure

The above algorithm summarizes the procedure for filling the A affinity matrix. At the beginning each element of the matrix is set to 0.0 and then if the similarity measure between the nodes is greater than a threshold, this measure becomes: $a_{ij} = \theta(d_j, d_i)$, where d_i and d_j represent two images of the dataset, that are projected in the same bucket for LSH kNN graph or in the same or 1-neighbour bucket for multi-probe LSH kNN graph approach.

Unfortunately, it is impossible to apply this approach on large datasets, because pre-allocating the entire dense matrix depends to the available RAM memory and it will hardly possible to execute on datasets of size greater of 100k images. Therefore, for this case, instead of working on a dense matrix, a sparse matrix is used.

Sparse matrices can be used to reduce the computational time and still obtain good results also on large datasets, because the affinity matrices typically contain a lot of zeros. For instance, on Oxford5k dataset the approximate kNN graph has only the 0.7% of the edges of the brute-force kNN graph.

Moreover, considering that the matrix is symmetric, only the upper or lower values of the matrix are needed. Therefore, the previous condition adopted in the procedure of LSH kNN graph can be changed in this way:

$$a_{ij} = \begin{cases} \theta(d_i, d_j) & \text{if } j \geq i \wedge \theta(d_i, d_j) \geq th \\ 0 & \text{otherwise} \end{cases}$$

If the column index is not greater than row index, the rows and the columns are swapped due to the symmetric properties of the affinity matrix.

Two different types of sparse matrix has been tested: Compressed Row Storage (CRS) format and Coordinate (COO) format [7]. The CRS sparse matrix is composed by three vectors: values (containing the values of the dense matrix different from zero); column indexes (containing the column indexes of the

elements contained in the values vector); and row pointers (containing the locations of the values vector that indicate the beginning of a new row). Instead, the COO sparse matrix is composed by three vectors: a vector representing the non-zero elements (the values), the row and the column coordinate of each value contained in the values vector. The second solution is simpler than the first to implement, but it requires more space on disk.

However, using hash tables, it happens that the same edge weight is inserted multiple times. Therefore, every time a new value is inserted in a CRS matrix, checking whether the value is already in the matrix might be a possible solution. Unfortunately, this tends to be a time consuming process. Conversely, using a COO matrix, all the values (including repeated ones) are inserted, but a sorting is performed and duplicates are removed. Applying once the sorting and removing the duplicates is faster than performing $\mathcal{N} \cdot L$ times the search, given that sorting has a $O(\mathcal{N} \log_2 \mathcal{N})$ complexity which is lower than the $O(\mathcal{N})$ complexity of the search.

References

1. Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms, pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)
2. Chen, J., Fang, H.r., Saad, Y.: Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection. Journal of Machine Learning Research **10**(Sep), 1989–2012 (2009)
3. Chum, O., Philbin, J., Sivic, J., Isard, M., Zisserman, A.: Total recall: Automatic query expansion with a generative feature model for object retrieval. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2007)
4. Debatty, T., Michiardi, P., Thonnard, O., Mees, W.: Building k-nn graphs from large text data. In: IEEE International Conference on Big Data, pp. 573–578. IEEE (2014)
5. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web, pp. 577–586. ACM (2011)
6. Douze, M., Szlam, A., Hariharan, B., Jégou, H.: Low-shot learning with large-scale diffusion. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3349–3358 (2018)
7. Golub, G.H., Van Loan, C.F.: Matrix computations, vol. 3. JHU press (2012)
8. Gordo, A., Almazan, J., Revaud, J., Larlus, D.: End-to-end learning of deep visual representations for image retrieval. International Journal of Computer Vision **124**(2), 237–254 (2017)
9. Houle, M.E., Ma, X., Oria, V., Sun, J.: Improving the quality of K-NN graphs for image databases through vector sparsification. In: Proceedings of International Conference on Multimedia Retrieval, p. 89. ACM (2014)
10. Huiskes, M.J., Lew, M.S.: The MIR flickr retrieval evaluation. In: Proceedings of the 1st ACM international conference on Multimedia Information Retrieval, pp. 39–43. ACM (2008)
11. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 604–613. ACM (1998)
12. Iscen, A., Tolias, G., Avrithis, Y., Chum, O.: Mining on manifolds: Metric learning without labels. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7642–7651 (2018)
13. Iscen, A., Tolias, G., Avrithis, Y.S., Furon, T., Chum, O.: Efficient diffusion on region manifolds: Recovering small objects with compact CNN representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, p. 3 (2017)
14. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(1), 117–128 (2011)
15. Li, D., Hung, W.C., Huang, J.B., Wang, S., Ahuja, N., Yang, M.H.: Unsupervised visual representation learning by graph-based consistent constraints. In: European Conference on Computer Vision, pp. 678–694. Springer (2016)
16. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: Proceedings of the 33rd international conference on Very Large Data Bases, pp. 950–961. VLDB Endowment (2007)

17. Magliani, F., Fontanini, T., Prati, A.: Efficient nearest neighbors search for large-scale landmark recognition. International Symposium on Visual Computing (2018)
18. Magliani, F., Fontanini, T., Prati, A.: Landmark recognition: From small-scale to large-scale retrieval. In: Recent Advances in Computer Vision, pp. 237–259. Springer (2019)
19. Magliani, F., Prati, A.: An accurate retrieval through R-MAC+ descriptors for landmark recognition. In: Proceedings of the 12th International Conference on Distributed Smart Cameras, p. 6. ACM (2018)
20. Magliani, F., Sani, L., Cagnoni, S., Prati, A.: Genetic algorithms for the optimization of diffusion parameters in content-based image retrieval. In: Proceedings of the 13th International Conference on Distributed Smart Cameras, p. 14. ACM (2019)
21. Park, Y., Park, S., Lee, S.g., Jung, W.: Scalable k-nearest neighbor graph construction based on greedy filtering. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 227–228. ACM (2013)
22. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2007)
23. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Lost in quantization: Improving particular object retrieval in large scale image databases. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2008)
24. Radenović, F., Iscen, A., Tolias, G., Avrithis, Y., Chum, O.: Revisiting oxford and paris: Large-scale image retrieval benchmarking. In: CVPR (2018)
25. Sieranoja, S., Fränti, P.: Fast random pair divisive construction of knn graph using generic distance measures. In: Proceedings of the 2018 International Conference on Big Data and Computing, pp. 95–98. ACM (2018)
26. Tolias, G., Sicre, R., Jégou, H.: Particular object retrieval with integral max-pooling of CNN activations. International Conference on Learning Representations (2016)
27. Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., Li, S.: Scalable k-nn graph construction for visual descriptors. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1106–1113. IEEE (2012)
28. Xu, J., Wang, C., Qi, C., Shi, C., Xiao, B.: Iterative manifold embedding layer learned by incomplete data for large-scale image retrieval. IEEE Transactions on Multimedia (2018)
29. Zhang, Y.M., Huang, K., Geng, G., Liu, C.L.: Fast kNN graph construction with locality sensitive hashing. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 660–674. Springer (2013)
30. Zhou, D., Weston, J., Gretton, A., Bousquet, O., Schölkopf, B.: Ranking on data manifolds. In: Advances in Neural Information Processing Systems, pp. 169–176 (2004)