An Efficient Hashing Algorithm for NN Problem in HD Spaces

Faraj Alhwarin(✉), Alexander Ferrein, and Ingrid Scholl

Mobile Autonomous Systems & Cognitive Robotics Institute, FH Aachen University of Applied Sciences, Eupener Straße 70, 52066 Aachen, Germany
{alhwarin,ferrein,scholl}@fh-aachen.de

Abstract. Nearest neighbor (NN) search is a fundamental issue in many computer applications, such as multimedia search, computer vision and machine learning. While this problem is trivial in low-dimensional search spaces, it becomes much more difficult in higher dimension because of the phenomenon known as the curse of dimensionality, where the complexity grows exponentially with dimension and the data tends to show strong correlations between dimensions. In this paper, we introduce a new hashing method to efficiently cope with this challenge. The idea is to split the search space into many subspaces based on a number of jointly-independent and uniformly-distributed circular random variables (CRVs) computed from the data points. Our method has been tested on datasets of local SIFT and global GIST features and was compared to locality sensitive hashing (LSH), Spherical Hashing methods (HD and SHD) and the fast library for approximate nearest neighbor (FLANN) matcher by using linear search as a baseline. The experimental results show that our method outperforms all state-of-the-art methods for the GIST features. For SIFT features, the results indicate that our method significantly reduces the search query time while preserving the search quality and outperforms FLANN for datasets of size less than 200 K points.

Keywords: Feature matching · Hash trees · NN search
Curse of dimensionality

1 Introduction

The nearest neighbor (NN) search is one of the most challenging tasks in many computer applications such as machine learning, multimedia databases, computer vision and image processing. In many of these applications, the data points are typically represented as high-dimensional vectors. For example, in computer vision applications, to automatically process and understand an image, it has to be described by one or several of high-dimensional features.

The NN Search problem is defined as follows: Assuming a dataset of points $P \subset \mathbb{R}^d$ are given as $P = \{p_1, p_2, p_3, \ldots, p_n\}$. Then, the problem is to find the

M. De Marsico et al. (Eds.): ICPRAM 2018, LNCS 11351, pp. 101–115, 2019.
https://doi.org/10.1007/978-3-030-05499-1_6

closest point in P to a given query point q using a certain similarity measure such as the hamming or euclidean distance.

$$NN(P,q) = \{p \in P \mid \forall p_i \in P \wedge p_i \neq p : dist(p,q) \leqslant dist(p_i,q)\}.$$

The linear search is the easiest technique to solve this problem, which includes computing all the metric distances from a query point to every point in the dataset for finding the point with the smallest distance (exact NN). However, the query time of linear search grows proportionally to the number and dimensionality of data points. Therefore, this solution is very time-consuming and impractical for large-scale datasets of high-dimensional points.

Much research has been conducted within the last three decades to find an efficient solution for the NN search problem in high-dimensional spaces. The suggested solutions have in common that they organise the dataset content in complex data structures (trees, graphs or hash tables) in such a way that a NN query can be answered without searching the whole dataset. Unfortunately, the time of data saving into and fetching from the data structure keeps growing exponentially with the number of dimensions. The NN search can be accelerated by relaxing the problem by searching for the approximate nearest neighbor (ANN) instead of the exact one. The ANN is defined as any point whose distance to query point is less than $(1+\varepsilon)$ times the distance between the query point and exact NN.

In general, the ANN search algorithms can be typically classified into three major categories: graph-based, tree-based, and hash-based algorithms [1].

The common idea of graph-based algorithms is to build a k-nearest neighbor (kNN) graph in an offline phase. The kNN graph is a network of nodes linked by weighted edges. The nodes represent the data points and edge weights represent the distances between linked points. In the literature, many strategies for kNN graph exploration have been published. In [2], the kNN graph is explored in a best-first order, starting from a few well-separated nodes. In [3], a greedy search is performed, to find the closest node to the query point, staring from a randomly selected starting node.

The tree-based algorithms are based on the recursive partitioning of the search space into sub-spaces.

The most widely used tree-based algorithm is the k-d tree [4,5]. The k-d tree is a k-dimensional binary search tree in which each node represents a partition of the k-dimensional space and the root node represents the whole space. To search for the NN, the coordinates of the query point are used to determine the NN leaf node. Then, the linear search is performed to determine the closest point within the NN leaf. The k-d tree operates successfully in low-dimensional search space, but its performance degrades exponentially with increasing number of dimensions. Various improvements to k-d tree have been suggested [6,7]. Silpa-Anan et al. [8] proposed the use of multiple randomised k-d trees, which are built by selecting the split dimensions randomly from among the dimensions with a high variance. By querying the randomised trees, a single priority queue is used to save answers sorted by their distances to the query point.

In [9], the hierarchical k-means tree is proposed. The hierarchical k-means tree partitions the space hierarchically by using the k-means clustering algorithm.

The k-means algorithm selects k points randomly (called centroids). Then, k initial clusters are created by clustering points according to their distances to the closest centroid. The centroids are iteratively updated to the mean of each cluster, and the clustering is repeated until convergence is achieved.

The most popular hash-based method for ANN search is the locality sensitive hashing (LSH). The LSH was introduced by Indyk et al. [10] for use in binary hamming spaces and later modified by Datar et al. [11] for the use in euclidean spaces. The main idea of LSH is to construct a set of hash functions that project data points from a high-dimensional to one-dimensional space, segmented into intervals of the same length (called buckets). The hash functions are designed, so that the neighbor points in the origin space will be projected to the same hash buckets with high probability [12]. The performance of hash-based algorithms highly relies on the quality of the hash functions used and the tuning of algorithmic parameters. Therefore, many papers dealing with these issues have been proposed (e.g. [13–19]).

Recently, in [20] Heo et al. proposed spherical hashing (SHD) method and compared it with many recent algorithms such as spectral hashing [19], iterative quantization [16], random maximum margin hashing [18] and generalized similarity preserving independent component analysis [17]. They found that their method outperforms all compared state of-the-art algorithms. In [1], Muja and Lowe compared many different algorithms for ANN search using datasets with a wide range of dimensionality. They developed a *Fast Library for ANN search* (FLANN) in high dimensional spaces. FLANN contains a collection of the best algorithms for ANN search and an automatic mechanism for electing the best algorithm and optimal parameters depending on the dataset's content.

In [21], we introduced a hashing method for ANN search in high dimensional spaces. The idea is based on extracting several jointly-independent and uniformly-distributed circular random variables (CRVs) from the data points. These CRVs are then used to index data in hash trees. The CRVs hashing (CRVH) method has been successfully applied to speed up SIFT feature matching. In this paper, we extend our previous work by applying the algorithm on GIST descriptor and by evaluating its performance on large image datasets.

The rest of the paper is organised as follows. In the next Sect. 2, we present CRVH method in more details. In Sect. 3, we show how to apply our method on SIFT and GIST descriptors. In Sect. 4, we evaluate our method on several datasets of SIFT and GIST descriptors (with different sizes) and compare it with the LSH, HD, SHD and FLANN methods. Finally, we conclude the paper in Sect. 5.

2 CRVH Method

In this section, the proposed CRVH method will be theoretically described in details. We start with defining the circular random variables (CRVs).

Definition 1. *Let* $\boldsymbol{x} = (x_0 \ \ldots \ x_{n-1})^T$ *be a n-dimensional descriptor vector. We segment it into k segments* $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{k-1}$*, each of length l with* $k = \lfloor \frac{n}{l} \rfloor$*. Each segment* $\boldsymbol{s}_i$ *is represented by a l-dimensional vector:* $\boldsymbol{s}_i = \{x_{i \cdot l}, x_{i \cdot l+1}, \ldots, x_{i \cdot l+(l-1)}\}$*. For each segment, a circular random variable* v_i *can be defined as the location of the maximum component within the segment.* $v_i = \{j \in [0, l-1] \mid x_{i \cdot l+j}$ *is the maximum value in* $\boldsymbol{s}_i\}$*.*

From CRVs, a hash tree with l^k leaves can be constructed. The hash keys $h(\boldsymbol{x})$ are determined by a polynomial of order $(k-1)$ as follows:

$$h(\boldsymbol{x}) = I = \sum_{i=0}^{k-1} l^i \cdot v_i. \tag{1}$$

Where l denotes the segment length and k the number of segments (the number of CRVs). If the CRVs are jointly-independent and uniformly-distributed, then the data points will be evenly distributed over all the hash tree leaves. In Fig. 1, we show how to extract CRVs from a high-dimensional descriptor. The descriptor is divided into k segments each of length $l = 5$. Form each segment $\boldsymbol{s}_i$, a CRV v_i is defined as the peak index within the respective segment.

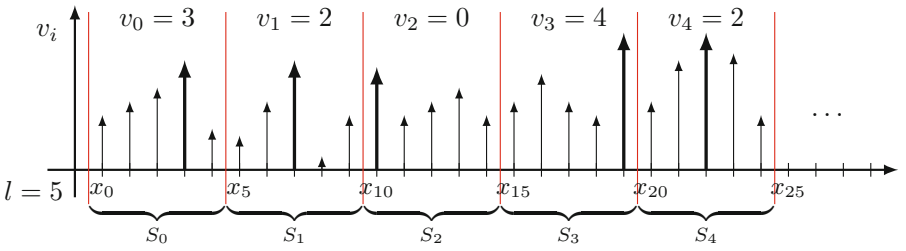


Fig. 1. Extraction of CRVs from a data point represented as d-dimensional vector. In this example, the segment length is chosen equal to 5 [21].

For two neighboring points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ it is assumed that the CRVs computed from its descriptors tend to be the same and both points are hence hashed into the same hash-tree leaf with high probability

$$Prob\left[h\left(p_1\right) = h\left(p_2\right)\right] > P_1$$
$$Prob\left[h\left(p_1\right) \neq h\left(p_2\right)\right] < P_2$$

with $P_1 \gg P_2$. P_1 is a threshold of the probability that two true neighbors are hashed to the same leaf and P_2 is a threshold of the probability that two true neighbors are hashed to different leaves. For the sake of simplicity, we explain this assumption in two-dimensional search space as shown in Fig. 2. In 2D space,

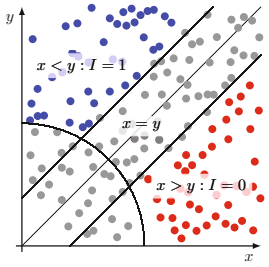


Fig. 2. 2D classification using one CRV [21]. (Color figure online)

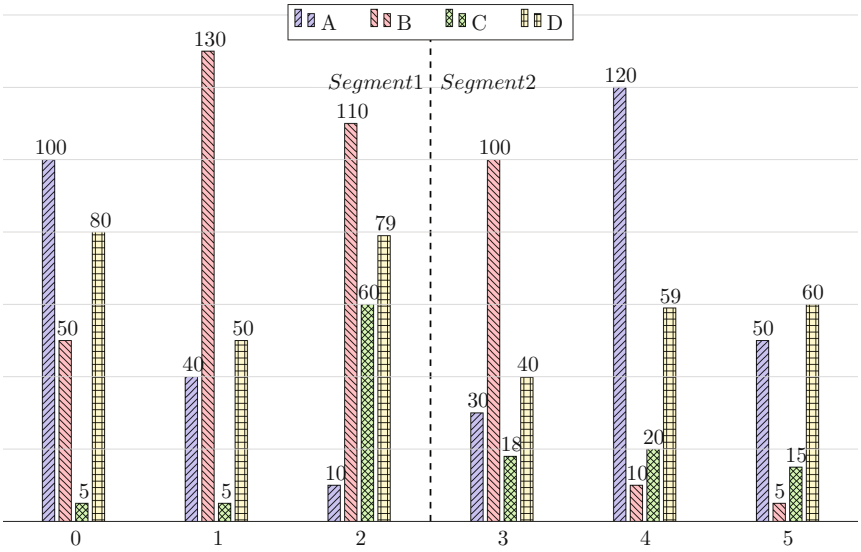


Fig. 3. Vectors of 6D points, A, B, C and D.

the vector can be considered as one segment of length $l = 2$ and hence only one circular binary random variable can be constructed v. If the abscissa of a vector is larger than the ordinate ($x > y$) we get $v = 0$ (red dots in Fig. 2), otherwise $y > x$ yields $v = 1$ (blue dots in the figure). For boundary points, which whose segments have no dominant maximum (grey dots in Fig. 2), we get a boundary problem. In this case, boundary problem can be avoided by adding boundary points to both hash leaves. In high-dimensional spaces, the boundary problem can be solved by considering not only the maximum indices that define the CRVs, but also the second maximum indices if the second to first maximum ratio is greater than a certain threshold T.

$$T = \frac{max_2}{max_1} \tag{2}$$

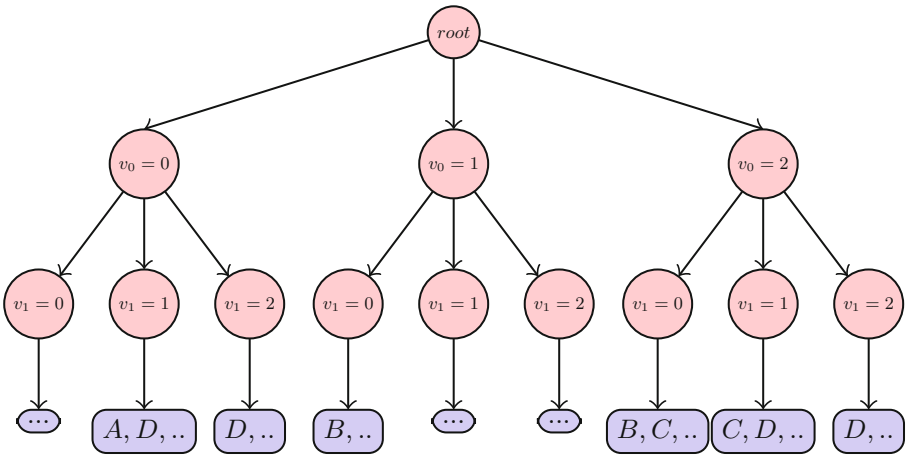


Fig. 4. Hash tree for 6D dataset with two CRVs of segment length = 3 [21].

where max_1 and max_2 are the first and second maximum values in a certain segment, respectively. The adjustment of ratio threshold is used to make a trade-off between search speed and precision. This consideration can be taken into account while storing or/and querying stages. The dealing with boundary problem can be explained by the following example: Assuming that we have 4 of 6D points represented in Fig. 3 and we chose segment length equal to 3, then we obtain two CRVs. Using these CRVs, points can be stored into a hash tree of $3^2 = 9$ leaves as shown in Fig. 4.

While querying, we can distinguish between three cases: first, each segment of the query point has a dominant peak as the case of point A (see Fig. 3), in this case, only one of the 9 leaves has to be searched. In second case, one of the segments has no dominant peak (as the case of points B or C). In this case, we have to consider the maximum and the second maximum and hence two leaves have to be searched. The last case is if both segments have no dominant peaks (as in the case of point D). In this case, 4 leaves out of 9 have to be explored. For illustration purposes, Fig. 3 shows the maximum, second maximum and their indices of two segments for 4 example points, and Fig. 4 shows how they are stored (or queried) in the hash tree, when the ratio threshold T is set to 0.5.

Using the above hash function, the dataset points will be evenly distributed over all the hash leaves, if two conditions are met, firstly the used CRVs are uniformly-distributed, and secondly, the CRVs are jointly-independent.

To verify if the CRVs meet the uniformly-distributedness condition, their probability density functions (PDFs) are estimated from a large set of data points. The PDFs are computed by constructing histograms of the CRVs ranging between $[0, l-1]$. Once the PDFs are estimated, the χ^2 test is used to quantitatively evaluate the goodness fit to the uniform distribution. The value of the test statistic is defined as:

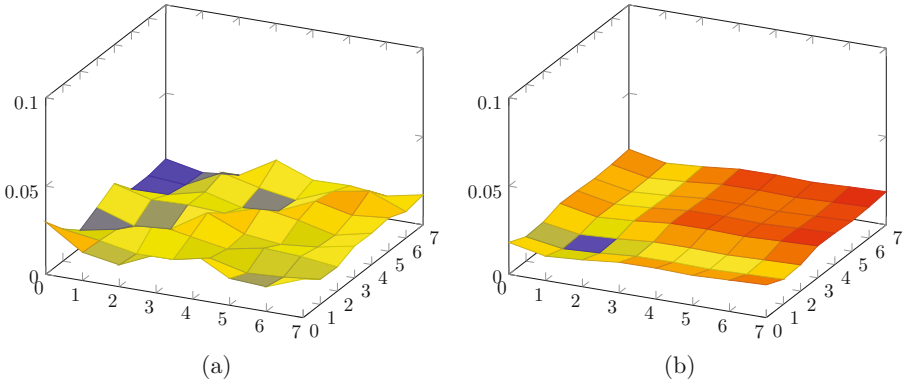


Fig. 5. Comparison between (a) joint pdf of two uncorrelated CRVs and (b) the product of their individual pdfs.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E)^2}{E} \tag{3}$$

where O_i is the observed pdf value, E is the expected value from the uniform distribution and n is the number of possible values the CRVs can take. Mathematically, it is known that, a set of random variables $(V_1, V_2, \ldots, V_n)$ are jointly-independent, if the joint probability density function is equal to the product of their individual pdfs.

$$pdf(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} pdf(V_i) \tag{4}$$

To verify whether the CRVs meet the jointly-independent condition, the circular correlation coefficient (CCC) is firstly used to filter out correlated CRVs. The jointly-independent condition is then examined for the remaining uncorrelated CRVs by comparing the joint probability density function with the product of the individual probability functions. In our case, we experimentally find that CCC is sufficient to determine whether the CRVs are jointly-independent or not. In Fig. 5, the joint PDF and the product of PDFs of two uncorrelated CRVs (extracted from one million 960-D GIST descriptors) are shown. As seen by comparing the Fig. 5(a) with Fig. 5(b), for two uncorrelated CRVs, the joint PDF tends to be equal to the individual PDFs'product.

The CCC proposed by Fisher and Lee [22] is defined as:

$$\rho(\alpha, \beta) = \frac{\sum_{i=0}^{n} \sin(\alpha - \bar{\alpha}) \cdot \sin(\beta - \bar{\beta})}{\sqrt{\sum_{i=0}^{n} \sin(\alpha - \bar{\alpha})^2 \cdot \sum_{i=0}^{n} \sin(\beta - \bar{\beta})^2}} \tag{5}$$

where $\rho(\alpha, \beta)$ is the CCC, n is the number of data points, α, β are two circular variables and $\bar{\alpha}$, $\bar{\beta}$ are their respective circular mean values defined as:

$$mean\,(\alpha) = \bar{\alpha} = \arctan\left(\sum_{i=0}^{n} \sin(\alpha), \sum_{i=0}^{n} cos\,(\alpha)\right). \tag{6}$$

the absolute of $\rho(\alpha, \beta)$ takes values from the interval $[0, 1]$. 0 indicates that there is no relationship between the variables, and 1 represents the strongest association possible.

Once the CCC and the χ^2 test values are computed for all the extracted CRVs, the CRVs are grouped so that CCC and χ^2 values are as small as possible in each group. An example with CCCs computed for SIFT descriptors is shown in Fig. 6. For each group of uncorrelated CRVs, a hash tree can be construct.

The above conditions ensure that the data are well-balancedly distributed over all the hash tree leaves. Therefore, the speed-up factor gained by our method compared to linear search can be theoretically determined as follows:

$$(\frac{l}{2})^k \leq SF \leq l^k \tag{7}$$

where l is CRV length and k is the number of CRVs that meet the conditions mentioned above.

In general, our method consists of three main steps. In the first step we extract the CRVs from the descriptor. To this end, we study the characteristics of the descriptor statistically to determine how its components are distributed and dependent on each other. Based on these characteristics we divide the descriptor into a set of equal length segments. For each segment, a CRV is defined as the location of the maximum within this segment. We calculate then the probability density functions and the joint dependency between the CRVs, and we group them into groups so that they are uniformly-distributed and jointly-independent as much as possible. The goal behind that is to spread the points over hash leaves uniformly and hence to maximize the speed-up factor defined in Eq. 7 regardless of how the points are distributed in their origin space.

In the second step we store the dataset points into hash trees. For this goal, we calculate the CRVs from the segments specified in the first step. From CRVs of each point, a hash key is computed as defined in Eq. 1. The hash key is then used to specify the hash leaves where to store this point. This step is done once for each dataset.

The last, third step is the same as the second one, but it is run on-line and applied to each query point in order to determine the hash leaves, where the candidate neighbors of the query point can be expected.

In the next section, we will describe how to apply this method on local SIFT and global GIST descriptors, which are mostly used in computer vision and image processing applications.

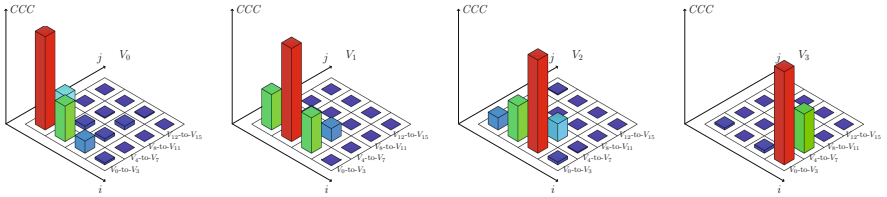


Fig. 6. Plot of circular correlation coefficients for V_0 to V_3 between each two CRVs for SIFT descriptors (out of 16) [21].

3 CRVH Method for SIFT and GIST Descriptors

To apply CRVH method on SIFT descriptors, we choose the CRV segment length $l = 8$. Then, from the 128-dimensional vector, 16 different CRVs can be obtained. A subset of these CRVs have to be selected, so that the CRVs meet the jointly-independent and uniformly-distribution conditions. For this goal, we have statistically analysed a dataset of the SIFT descriptors. Statistically, we found that the SIFT descriptor has a special signature, so that some components are always larger than some others [21]. The signature of SIFT descriptors is represented in Fig. 7 by the mean value of each component. For example, the 41^{th}, 49^{th}, 73^{th} and the 81^{th} components are always significantly larger than their neighbors.

The signature of SIFT descriptors influences the distribution of proposed CRVs. In order to remove this effect, SIFT descriptors are weighted before computing of CRVs. The weight vector is defined as the inverse of individual elements of the signature vector. $S = [s_1, s_2, \ldots, s_n] \Rightarrow W = \left[s_1^{-1}, s_2^{-1}, \ldots, s_n^{-1}\right]$ Fig. 8 shows the PDF functions of the CRVs before and after removing the descriptor signature effect. As shown in Fig. 8(b), after removing signature effect, all CRVs meet the uniformly-distributedness condition.

To study the dependence between the CRVs, the circular correlation coefficient (CCC) between each two CRVs is calculated. The CCCs between CRVs are explained in Fig. 6. Figure 6 shows that neighboring CRVs in the descriptor are highly-correlated, whereas there are no or only very weak correlations between non-contiguous CRVs. We omitted the other 12 diagrams showing the correspondences for the other CRVs. From the 16 CRVs we get two sub-groups of jointly-independent and uniformly-distributed CRVs:

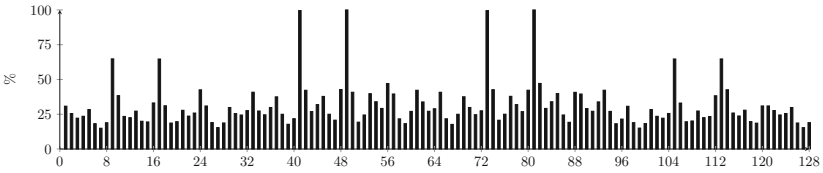
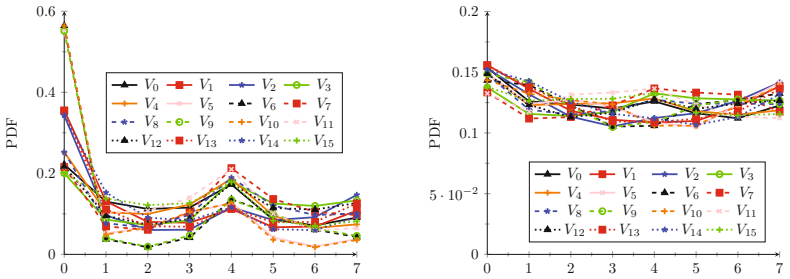
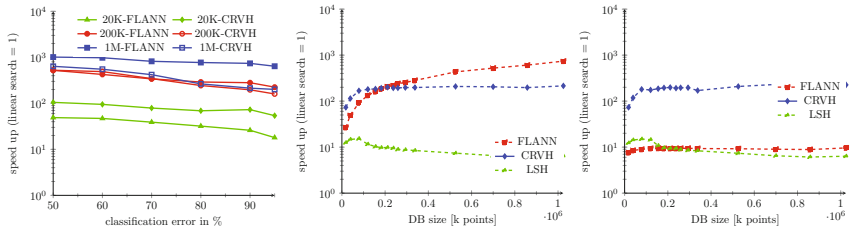


Fig. 7. The signature of a SIFT descriptor; the normalised mean values of the SIFT descriptor components were computed from a dataset of 100 K descriptors.



(a) PDFs of CRVs (b) PDFs of CRVs after removing signature effect

Fig. 8. The probability density functions of the CRVs (extracted from 100k SIFT features) before and after removing signature influence.



(a) Trade-off between speed-up and classification precision for different dataset sizes. (b) Static dataset (c) Dynamic dataset

Fig. 9. Speed-up and precision comparison between CRVH and FLANN; baseline is linear search for static and dynamic datasets.

$g_1 = \{V_0, V_2, V_5, V_7, V_8, V_{10}, V_{13}, V_{15}\}$
and $g_2 = \{V_1, V_3, V_4, V_6, V_9, V_{11}, V_{12}, V_{14}\}$.
From these two sub-groups, two hash trees can be constructed.

With a segment length of 8, 120 or 48 different CRVs can be obtained from 960-D or 384-D GIST descriptors respectively. These CRVs are classified into several sub-groups, so that the CRVs of each group have to meet the jointly-independent and the uniformly-distribution conditions. Similar to SIFT, we statistically found that a GIST descriptor has also a special signature, so that some components are always larger than some others. Before computing CRVs from the GIST descriptor, this signature is neutralised by weighting the descriptor with a constant weight vector. The weight vector is computed as the case of SIFT feature by inverting the signature components. Experimentally, we found that, CRVs of 960-D GIST descriptor can be classified into 14 sub-groups, so that each sub-group contains 6 jointly-independent and uniformly-distributed CRVs. Hence, from these sub-groups, 14 hash trees can be constructed. Similarly, for 384-D GIST descriptor, 6 hash trees can be constructed.

4 Empirical Evaluation

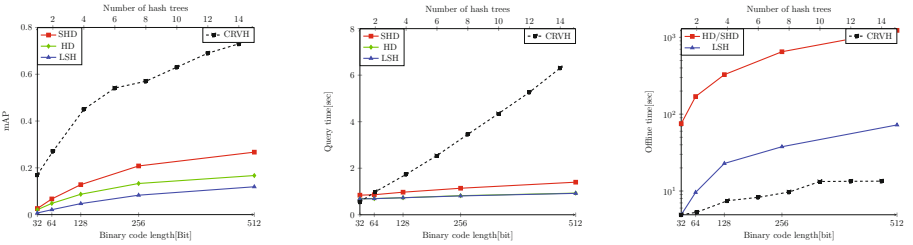
In this section, we analyse the performance of our method on two kinds of descriptors: local SIFT and Global GIST descriptors. We further compare the method with FLANN, LSH, DH and SHD on different image datasets. In the following experiments, we use the OpenCV implementation of FLANN and the C++ implementation of LSH, HD and SHD available in [20]. All experiments were carried out on a Linux machine with an Intel(R) Core(TM) i7-4770S CPU 3.10 GHz and 32 GB RAM.

For SIFT descriptors, the performance of our method is compared with the state-of-the-art NN matcher FLANN and LSH. The experiments are carried out using the *Oxford Buildings Dataset* of real-word images [23]. The *Oxford Buildings Dataset* dataset consists of about 5000 images. Among them there are several pairs that show the same scene from different viewpoints. 10 images of that pairs are taken out of the dataset and used as the query set.

We compare our method with FLANN and LSH in terms of both, speed-up over the linear search (baseline) and the percentage of correctly sought neighbors (precision). To evaluate the performance of our method, two experiments were conducted. The first experiment was carried out with different dataset sizes (20 K, 200 K, 1M) by varying precision parameters. We measured the trade-off between the speed-up and the precision. For the FLANN matcher, the precision was adjusted by varying appropriate FLANN parameters (number of trees and checks), whereas for the CRVH method, the precision was changed by varying the ratio threshold. The obtained results are shown in Fig. 9(a). As can be seen from the figure, our method outperforms the FLANN matcher for datasets with a size of less than 200 K descriptors.

In the second experiment, the performance is compared against FLANN and LSH for two different settings, a static and a dynamic dataset, respectively. In the static setting, the image dataset remains unchanged, while in the dynamic one, the dataset needs to be updated on-line by adding or deleting images. In this experiment, we keep the precision level at 90% and vary the size of dataset. Figure 9 shows the obtained results for both dataset settings. Figure 9(c) shows that in the case of a dynamic datasets, the CRV method outperforms both the LSH and FLANN methods for all dataset sizes significantly. It reaches speed-up factor of 20 over FLANN for dataset sizes up 100 K descriptors. The reason of this outcome can be explained by the FLANN method constructs a specific nearest neighbor search index for a specific dataset; when the dataset is updated by adding or removing some data, the search index has to be updated as well, otherwise the search speed decrease. Conversely, the CRVH method works independently from the dataset contents and its performance is not influenced by adding or removing data points.

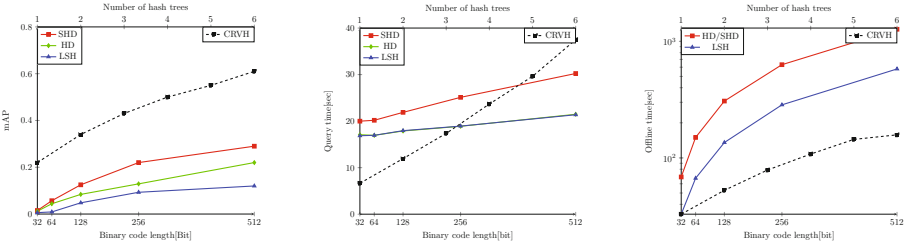
For the GIST descriptor, the performance of our method is compared with LSH and Spherical Hashing methods (HD and SHD). The experiments are carried out using the Tiny [24] dataset. The Tiny dataset consists of 80 million 32×32 color images. Each image is described by one GIST descriptor of dimension 960 or 384. We evaluate our method on three different-sized sub-sets of



(a) mAP (b) Query time (c) Offline time

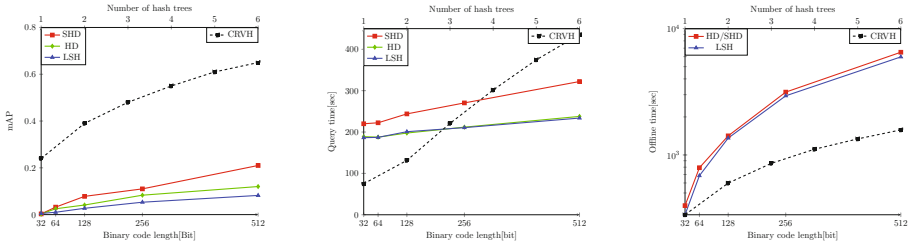
Fig. 10. Results on 50 k 960-D GIST descriptors.

the Tiny dataset (50K, 1M and 10M images). In each experiment, 100 GIST descriptors are randomly selected and used as query set. The remaining descriptors are used as the dataset. To obtain statistically meaningful results, for 50 K and 1M datasets, the experiments were repeated 5 times with varying the data- and query sets. For 10M, the experiment was repeated only two times because of the time required for computing ground truth. The performance is measured by mean average precision (mAP), query time and offline time. The precision is defined as the fraction of true points among the top Knn retrieved points. The ground truth is determined by the Knn nearest neighbors that are retrieved by the linear search based on Euclidean distance of GIST descriptors. For the LSH, HD, and SHD hashing methods, the performance was measured across code binary lengths ranging from 32 to 512 bits, whereas for CRVH, the performance was measured across all possible number of hash trees (14 trees for 960D and 6 trees for 384D GIST descriptors). For all experiments, we set $Knn = 100$, and we set the ratio threshold T of CRVH to $T = 0.5$. The offline time of LSH is the time required to compute binary code, while for HD/SHD it is the time of binary code computation plus the spherical hashing learning time. For our method, offline time is the time required to construct hash trees.



(a) mAP (b) Query time (c) Offline time

Fig. 11. Results on 1 million 385-D GIST.



(a) mAP (b) Query time (c) Offline time

Fig. 12. Results on 10 million 384-D GIST descriptors.

Figure 10 shows the results on 50 K 960D GIST descriptors. Using two hash trees, our method reaches a mAP (Fig. 10(a)) better than all state-of-the-art methods with a similar query time (see Fig. 10(b)). When increasing the number of threes, our method can reaches $mAP > 0.5$ with a query time less than 6 sec. Regarding offline time, Fig. 10(c) shows that our method extremely outperforms the other methods. For example, for a binary code length of 512, HD and SHD needed about 1200 s, while constructing 14 hash tree only needs about 15 s.

Figures 11 and 12 show the results on 1M and 10M of 384D GIST descriptors. For 3 hash trees, our method reaches $mAP = 0.45$ at query time about 15 s (Fig. 11(a)), while the best state-of-the-art method (SHD) reaches $mAP = 0.22$ at a query time about 30 s. Figures 11 and 12 also demonstrate the scalability of our methods. It is shown that, on both dataset sizes (1M and 10M), our method provides similar mAP, while the query and offline time increases linearly.

5 Conclusions

In this paper, we presented a hashing method for NN search in high-dimensional spaces. Our method bases on extracting a set of CRVs from data points. The data vector is divided into several segments. For each segment, a CRV is defined as the relative position of the peak in that segment. The CRVs are grouped together in such a way that in each group, they are all jointly-independent and uniformly-distributed. The CRVs are exploited to store data points evenly in one or several hash trees. In the query phase, the CRVs of query point are used to determine the hash leaves, where candidate neighbors can be found. The proposed method was tested on datasets of SIFT and GIST descriptors and compared with LSH, HD, SHD and FLANN. The presented experimental results show that, our proposed method outperforms all compared state-of-art methods.

References

1. Muja, M., Lowe, D.: Fast approximate nearest neighbors with automatic algorithm configuration. In: Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP 2009), pp. 331–340 (2009)
2. Sebastian, B., Kimia, B.B.: Metric-based shape retrieval in large databases. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2002), vol. 3, pp. 291–296 (2002)
3. Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., Zhang, H.: Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 1312–1317 (2011)
4. Bentley, L.: Multidimensional binary search trees used for associative searching. Commun. ACM (CACM) **18**(9), 509–517 (1975)
5. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw. **3**(3), 209–226 (1977)
6. Beis, J.S., Lowe, D.G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 1997), pp. 1000–1006 (1997)
7. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM **45**(6), 891–923 (1998)
8. Silpa-Anan, C., Hartley, R.: Optimised KD-trees for fast image descriptor matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2008), pp. 1–8 (2008)
9. Fukunaga, K., Narendra, P.M.: A branch and bound algorithm for computing k-nearest neighbors. IEEE Trans. Comput. (TC) **C–24**(7), 750–753 (1975)
10. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Symposium on Computational Geometry (SoCG 1998), pp. 604–613 (1998)
11. Datar, M., Indyk, P., Immorlica, N., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of computing (STOC 2004), pp. 253–262 (2004)
12. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117–122 (2008)
13. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: Proceedings of the IEEE 12th International Conference on Computer Vision (ICCV 2009), pp. 2130–2137 (2009)
14. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2010), pp. 3424–3431 (2010)
15. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: Proceedings of the International World Wide Web Conference (WWW 2005), pp. 651–660 (2005)
16. Gong, Y., Lazebnik, S.: Iterative quantization: a procrustean approach to learning binary codes. In: CVPR (2011)
17. He, J., Radhakrishnan, R., Chang, S.-F., Bauer, C.: Compact hashing with joint optimization of search accuracy and time. In: CVPR (2011)
18. Joly, A., Buissonr, O.: Random maximum margin hashing. In: CVPR (2011)
19. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2008)

20. Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.E.: Spherical hashing: binary code embedding with hyperspheres. IEEE Trans. Pattern Anal. Mach. Intell. **37**, 2304–2316 (2015)
21. Alhwarin, F., Ferrein, A., Scholl, I.: CRVM: circular random variable-based matcher, a novel hashing method for fast NN search in high-dimensional spaces. In: Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2018), pp. 214–221 (2018)
22. Fisher, N.I., Lee, A.: A correlation coefficient for circular data. Biometrika **70**(2), 327–332 (1983)
23. 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 (CVPR 2007) (2007)
24. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: a large data set for nonparametric object and scene recognition. IEEE Trans. Pattern Anal. Mach. Intell. **30**(11), 1958–1970 (2008)