QUERY-ADAPTATIVE LOCALITY SENSITIVE HASHING

Hervé Jégou[1], *Laurent Amsaleg*[2], *Cordelia Schmid*[1] *and Patrick Gros*[3]

[1]INRIA Grenoble, LJK ; [2]CNRS, IRISA ; [3]INRIA Rennes, IRISA

ABSTRACT

It is well known that high-dimensional nearest-neighbor retrieval is very expensive. Many signal processing methods suffer from this computing cost. Dramatic performance gains can be obtained by using approximate search, such as the popular Locality-Sensitive Hashing. This paper improves LSH by performing an on-line selection of the most appropriate hash functions from a pool of functions. An additional improvement originates from the use of E_8 lattices for geometric hashing instead of one-dimensional random projections. A performance study based on state-of-the-art high-dimensional descriptors computed on real images shows that our improvements to LSH greatly reduce the search complexity for a given level of accuracy.

***Index Terms*—** Search methods, Image databases, Quantization, Database searching, Information retrieval.

1. INTRODUCTION

Nearest neighbor search is inherently expensive due to the *curse of dimensionality* [1, 2]. This operation is required in many applications. In quantization, for example, it is used when assigning vectors to unstructured codebooks. In image retrieval or object recognition, the numerous descriptors of an image have to be matched with those of a descriptor dataset (direct matching) or a codebook (in bag-of-features approaches). Approximate nearest-neighbor (ANN) algorithms have been shown to be an interesting way of dramatically improving the search speed, and are often a necessity. Several *ad hoc* approaches have been proposed for vector quantization (see [3] for references), when finding the exact nearest neighbor is not exactly necessary, as long as the reconstruction error is limited. More specific ANN approaches performing content-based image retrieval using local descriptors have been proposed [4, 5]. Overall, one of the most popular approximate nearest neighbor search algorithms is the Euclidean Locality-Sensitive Hashing (LSH) [6, 7]. LSH has been successfully used for local descriptors [8] and 3D object indexing [7, 9].

In this paper, we build upon LSH to further improve its performance. Our main contribution is the following. First, we define a criterion that measures the expected accuracy of a given hash function with respect to the queries. At query time, this allows us to select on-line from a pool of hash functions the most appropriate ones. This improves the accuracy of the search. As in [10], a lattice replaces random projections, but the E_8 lattice [11, 12] is preferred over the Leech lattice as it is much cheaper to compute. Overall, our improvements turn the LSH into a query-adaptive locality sensitive lattice-based hashing that we call QA-LSH. We demonstrate the improvement over LSH in the context of searching high dimensional vectors, namely the state-of-the-art 128-dimensional SIFT [4] local descriptors, obtained from a set of images.

2. BACKGROUND

This section presents the background for LSH and E_8 lattices required to understand the remainder of this paper.

2.1. Locality-Sensitive Hashing

Indexing d-dimensional descriptors with the Euclidean version of LSH [7] proceeds as follows. The n vectors of the codebook are first projected onto a limited set of m directions characterized by the d-dimensional vectors $(a_i)_{1 \leq i \leq m}$ of norm 1. Each direction a_i is randomly drawn from an isotropic random generator.

For a given vector x and for each direction i represented by its vector a_i, a real-valued hash function $h_j^r(x)$ is defined as

$$h_i^r(x) = \frac{\langle x | a_i \rangle - b_i}{w}, \tag{1}$$

where w is the quantization step chosen according to the data (see [7]), the offset b_i is uniformly generated in the interval $[0, w)$ and where the inner product $\langle x | a_i \rangle$ is the projected value of the vector onto the direction i. The hash function h_i^r produces a real value which is subsequently rounded to an integer, as

$$h_i(x) = \lfloor h_i^r(x) \rfloor . \tag{2}$$

The family of hash functions $\mathcal{H} = \{h_i\}_{1 \leq i \leq m}$ is adapted to the case where the distance between vectors is Euclidean. If two vectors x and y are such that $h_i(x) = h_i(y)$, y is close to x with a good probability. Conversely, if x and y are far from each other, then $h_i(x) \neq h_i(y)$ with a high probability.

A single hash function of $\mathcal{H}$ is not discriminative enough by itself, because only one direction among d is used to partition the vectors. Therefore a second level of hash functions, based on the family $\mathcal{H}$, is subsequently defined. This level is formed by a family of l functions constructed by concatenating several functions from $\mathcal{H}$. Hence, each function g_i of this family is defined as

$$g_j = (h_{j,1}, \ldots, h_{j,k}), \tag{3}$$

where the functions $h_{j,i}$ are randomly chosen from the set $\mathcal{H}$ of hash functions. At this point, a vector x is indexed by a set of l vector of integers $g_j(x) = (h_{j,1}(x), \cdots, h_{j,k}(x)) \in \mathbb{Z}^k$.

The next step stores the vector identifier within the cell associated with this vector value $g_j(x)$. Since most of the cells are empty, a universal hash function u_1 is used to obtain from the vector $g_j(x)$ an integer lying in the interval $[0, c)$. The quantity c should be high enough to avoid, with high probability, the collision of two distinct integer vectors. For a vector $y = y_1, \cdots, y_k$, the function u_1 is defined [7] as

$$u_1(y) = \left(\left(\sum_{i=1}^{k} r'_i a_i \right) \mod P \right) \mod c, \tag{4}$$

where P is a prime (e.g., $P = 2^{32} - 5$) and r'_i are random integers. The integer $u_1(g_j(x))$ indexes a bucket in the hash table associated with the hash function g_j.

To further reduce the collision probability (e.g., if c is chosen small to reduce the memory usage), a second hash function u_2 similar to u_1 can be used (see [7]):

$$u_2(y) = \left(\sum_{i=1}^{k} r''_i a_i \right) \mod P, \tag{5}$$

where the r''_i are random integers different from the r'_i. In that case, the integer $u_2(g_j(x))$ is stored together with the vector identifiers in the bucket indexed by $u_1(g_j(x))$.

At run time, the query vector q is treated in the same way as the database elements, i.e., it is projected onto each of the l random lines, producing a set of l integer vectors $\{g_1(q), \cdots, , g_l(q)\}$. The algorithm keeps track of the vector identifiers encountered in the l buckets associated with the query, returning a short-list of vector identifiers. The nearest neighbor is found by performing an exact search within this short-list. For big datasets, this last step is the bottleneck in the algorithm.

As the benefits of using LSH are important for large datasets only, its efficiency will be measured by the number of vectors in the short-list for which we have to perform an exact search, i.e., the efficiency is assumed to be a linear function of the short-list length. Note, however, that in practice we have to keep in mind that the first steps of the algorithm may be slow if the parameters are set too large, in particular the number of hash functions l.

2.2. The E_8 lattice

Lattices have been widely studied in mathematics and physics, but some lattices have also been shown to be of high interest in quantization [3, 11]. For a uniform distribution, this gives better performance than any scalar quantizer [3]. Moreover, in contrast to unstructured sets of points, finding the nearest lattice point of a vector can be performed with an algebraic method [13]. This problem is referred to as *decoding*, due to the application in compression.

A lattice is a discrete subset of $\mathbb{R}^d$ defined by a set of vectors of the form

$$\{x = u_1 a_1 + \cdots + u_d a_d | u_1, \cdots, u_d \in \mathbb{Z}\} \tag{6}$$

where the vectors $a_1, \cdots, a_d$ are some linearly independent vectors of $\mathbb{R}^{d'}$, $d' \leq d$. In this paper, we will only consider lattices such that $d' = d$. Hence, denoting by $A = [a_1 \cdots a_d]$ the matrix whose columns are the vectors a_j, the lattice is the set of vectors spanned by Au when $u \in \mathbb{Z}^d$. With this notation a point of a lattice is uniquely identified by the integer vector u. Note that taking the diagonal matrix for A amounts to defining a regular grid of hypercubes.

In this section, we will on focus the hyper-diamond lattice E_8 which offers the best quantization performance for uniform 8-dimension vectors. As Conway and Sloane [12] point out, "It is worth drawing attention to the remarkably low value of the mean-squared error for E_8". Although the Leech lattice offers still slightly better quantization performance, our choice is motivated by the fact that the Leech lattice decoding requires significantly more operations, as for most high-dimensional lattices [13].

The E_8 lattice can be defined based on another 8-dimensional lattice, the D_8 lattice, which is the set of point of $\mathbb{Z}^8$ whose sum is even, e.g. $(1, 1, 1, 1, 1, 1, 1, 1) \in D_8$ whereas $(0, 1, 1, 1, 1, 1, 1, 1) \notin D_8$. Denoting by $\mathbf{\frac{1}{2}}$ the vector $(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$, the E_8 lattice is defined as the union

$$E_8 = D_8 \cup (D_8 + \mathbf{\frac{1}{2}}), \tag{7}$$

where $D_8 + \mathbf{\frac{1}{2}}$ is the set of points of D_8 translated by the vector $\mathbf{\frac{1}{2}}$, in particular $\mathbf{\frac{1}{2}} \in D_8 + \mathbf{\frac{1}{2}}$.

The decoding of a given vector x for the lattice E_8 can be done by performing the decoding for both the lattice D_8 and the lattice $D_8 + \mathbf{\frac{1}{2}}$. The decoding of x for D_8 proceeds as follows [12]:

- the nearest integer vector of x is found by rounding the components of x to the nearest integers;
- if the sum of this integer vector is even, then this vector is the nearest point of the lattice D_8;
- otherwise the second nearest integer vector is searched. This is done by searching for the component of x which is the farthest from an integer. This value is then rounded in the "bad way" [12] instead of the best rounding.

For example, if $x = (1.2, 1.2, 1.2, 1.2, 1.2, 1.1, 1.8, 1.4)$, then the closest integer vector is $(1, 1, 1, 1, 1, 1, 2, 1)$. Since its sum is odd, it does not belong to D_8. Hence, x is decoded as $(1, 1, 1, 1, 1, 1, 2, 2)$, since 1.4 is the component to be rounded in the bad way.

The decoding for $D_8 + \mathbf{\frac{1}{2}}$ is equivalent to decode the vector $x - \frac{1}{2}$ for the D_8 lattice and to add $\frac{1}{2}$ to the resulting vector. E.g., $x - \frac{1}{2} = (0.7, 0.7, 0.7, 0.7, 0.7, 0.6, 1.3, 0.9)$ is decoded as $(1, 1, 1, 1, 1, 1, 1, 1)$, giving $(1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5)$ for the lattice $D_8 + \mathbf{\frac{1}{2}}$.

The final decoding step for E_8 consists in comparing the distance between x and the closest vectors obtained from D_8 and $D_8 + \mathbf{\frac{1}{2}}$ and in retaining the closest one. Hence, in our example, x is decoded as $E_8(x) = (1, 1, 1, 1, 1, 1, 2, 2)$. The overall decoding requires 104 operations [14], which is much lower than the 3595 floating point operations needed for the Leech lattice [15]. Note that more efficient algorithms exist for the E_8 lattice [14] (about 20 operations). The algorithm of Conway and Sloane has been chosen for ease of presentation. Note also that there exists a suboptimal decoding algorithm for the Leech lattice that requires 519 operations [16].

3. QUERY-ADAPTIVE LSH (QA-LSH)

In this section, we first show that the probability of a given hash function returning the correct nearest neighbor is strongly correlated with a criterion that can be efficently computed without parsing the bucket. By exploiting this criterion and by using E_8 lattices as hash functions instead of random projections, we define an enhanced variation of the LSH, referred to as QA-LSH, which outperforms the original version in terms of the trade-off between retrieval accuracy and complexity.

3.1. Hash function relevance criterion

Let us consider the toy example in Fig. 1. The area under the curve corresponds to the probability $P(h_i(x) \neq h_i(NN(x)))$ that a projected query descriptor $h_i^r(x)$ is not quantized in the same bucket as its projected nearest neighbor $h_i^r(NN(x))$. For the sake of illustration, we assume here that the distance between the projected query descriptor and its projected nearest neighbor is Gaussian.

The figure illustrates the intuition that the position of the projection of x within the interval of size w has a strong impact on the probability that two vectors which are close are hashed in the

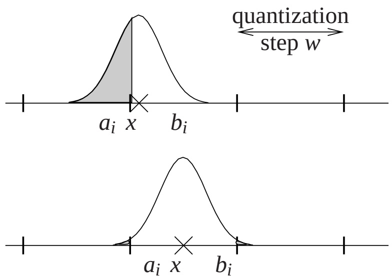

Fig. 1. Toy example: probability that both a descriptor and its noisy version are quantized in the same bucket under a Gaussian noise assumption.

same bucket. This position can be defined as the absolute value $|h_i^r(x) - (h_i(x) + 0.5)|$, which lies between 0 (the projected value is centered) and 0.5. This quantity gives some information on the relevance of the hash function used: lower values are better.

For a given query x, this information is used to define a relevance criterion $\lambda(g_j)$ for the hash functions g_j. Let us consider the real and integer hash functions $h_{j,i}^r$ and $h_{j,i}$ associated with g_j. The criterion is defined as

$$\lambda(g_j) = \sqrt{\sum_k (h_{j,i}^r(x) - (h_{j,i}(x) + 0.5))^2}. \qquad (8)$$

Under the assumption that the directions $a_{j,i}$ associated with the hash functions $h_{j,i}$ form an orthonormal basis[1] of the k-dimensional subspace onto which the dataset vectors are projected, the quantity $\lambda(g_j)$ corresponds to the distance of the projected vector from the center of the k-dimensional Voronoi cell. Note that $0 \leq \lambda(g_j) \leq \sqrt{k}/2$.

Fig. 2 shows the interest of the proposed criterion to measure the relevance of the different hash functions. It was generated using a set of SIFT descriptors randomly extracted from a large set of images. Given a hash function g_j, it shows the probability that a given SIFT descriptor is hashed in the same bucket as its nearest vector in the dataset. For a given query, this suggests choosing the hash functions $g_j(x)$ that provide the best expected relevance. It should be compared with Fig. 3, which gives the probability distribution function (PDF) of the proposed criterion on the same dataset. From these figures one can deduce that most of the hash functions used to retrieve the nearest neighbor have a low relevance.

3.2. Lattices as hash functions

The space partition resulting from the random projections is improved by using quantizers which are better suited to the Euclidean distance. Lattices are a natural choice, as they offer very good quantization properties for the mean square error dissimilarity measure used in Euclidean spaces. This has recently motivated the use of Leech lattices to perform the geometric hashing in LSH [10]. However, as explained in subsection 2.2, the interest of this Lattice in this context is mainly theoretical, as its decoding has a high computational cost. Therefore, this lattice is of interest only if the dataset

[1]Note that for d high enough the inner product between two isotropically drawn unitary vectors x and y is fairly approximated by a normal law of zero-mean and variance $1/d$. Hence, for $d = 128$, two directions a_{i_1} and a_{i_2} are almost orthogonal with very high probability.

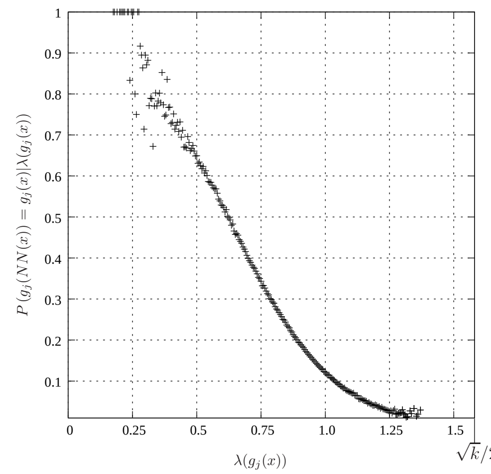

Fig. 2. Empirical probability that a SIFT descriptor (of dimension 128) is hashed into the same bucket as its nearest neighbor in a vector set comprising 50 000 elements ; $w = 0.2, k = 10$.

to be searched is extremely large, otherwise the lattice computation step is the bottleneck in the search.

We propose using the E_8 lattice instead, which offers excellent quantization properties together with a low decoding cost, as shown in subsection 2.2. For each hash function we randomly draw 8 components of a given input vector x, producing a 8-dimensional vector $x_{i,8}$. The hash functions of Eqn. 2 are then replaced by

$$h_i(x) = E_8\left(\frac{x_{i,8} - b_i}{w}\right), \qquad (9)$$

where $E_8(\cdot)$ represents the decoding function associated with the lattice E_8, b_i is a 8-dimensional random offset and w is a normalization factor defining the lattice cell size via a homothetic transformation. In contrast to random projections, $h_i(x)$ is an integer vector, not a scalar value. The construction of the functions g_j in Eqn. 3 is still valid, but here the g_j are the concatenation of the integer vectors h_i.

The square of the relevance criterion $\lambda(g_j)$ of Eqn. 8 is easily derived, by summing the square distances between the different vectors $h_{j,i}$ and their corresponding lattice points. Note that this quantity is obtained as a by-product of the E_8 lattice decoding algorithm.

3.3. The algorithm QA-LSH

The core ideas behind our QA-LSH algorithm are 1) using E_8 lattices as hash functions and 2) selecting the most appropriate hash functions using the proposed relevance criterion.

For this purpose, at query time we select the l' hash functions with the highest relevance for the query x, i.e. the ones minimizing $\lambda(g_j(x))$, instead of using all the hash functions g_j. Only the buckets associated with these l' hash functions will be parsed and the corresponding vectors retrieved. Therefore, quite a large set of l hash functions $\{g_i\}$ can be used. The main limitation is the amount of memory required to store the hash tables. The selection of the best hash functions is not a time consuming process, as the relevance criterion is obtained as a by-product of the lattice decoding. For a reasonable number of hash functions, the bottleneck of the algorithm remains the last step of the "exact" LSH algorithm, i.e., the search for the exact nearest neighbors within the short-list obtained

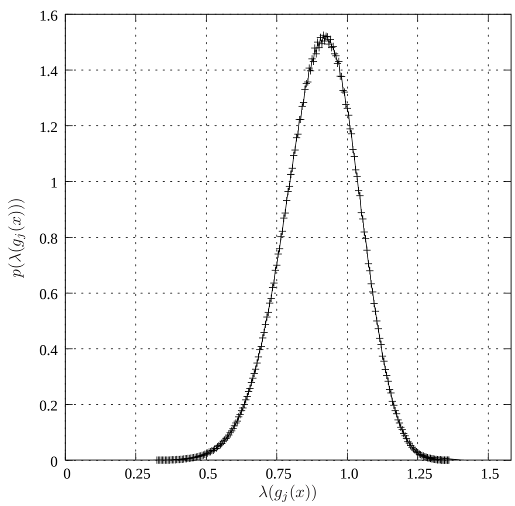

Fig. 3. Empirical PDF of the relevance criterion $\lambda(g_j(x))$; $w = 0.2, k = 10$. Note that for such a density, the value may be greater than 1, as for a normal distribution of variance lower than $\frac{1}{2\pi}$.

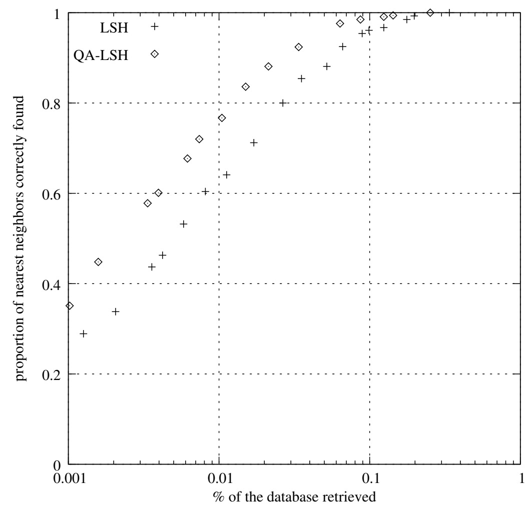

Fig. 4. Comparison of our method QA-LSH with E2LSH for the assignment of SIFT descriptor to a dataset of SIFT descriptors.

by parsing the buckets. Using E_8 lattices instead of random projections only impacts the hash function calculation: the rest of the algorithm remains identical.

4. SIMULATION: QA-LSH *vs* LSH

This simulation compares our QA-LSH algorithm with standard E2LSH [10] as described in Subsection 2.1. For this purpose, we measure the proportion of 128-dimensional SIFT descriptors (randomly extracted from a large set of images) correctly assigned to their exact closest element in a descriptor set composed of $k = 50\,000$ vectors. The objective is to measure the percentage of dataset elements for which an exact search has to be performed to obtain a given level of accuracy. This measure accurately reflects the computational cost for very large datasets, where the most time-consuming task is the exact search within the subset returned by the algorithm.

Fig. 4 shows that our new approach clearly outperforms the original LSH in terms of the trade-off between accuracy and efficiency measured by the number of vectors returned in the short-list. Note that for both algorithms, only the results associated with the sets of parameters (w and k) that perform well have been displayed.

5. CONCLUSION

We have presented an improved version of the popular approximate nearest neighbor search algorithm LSH. Our version uses the lattice structure and on-line selection of the quantization cells, leading to a better compromise between speed and accuracy.

6. REFERENCES

[1] C. Böhm, S. Berchtold, and D. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[2] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Intl. Conf. on Database Theory*, 1999, pp. 217–235.

[3] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. on Information Theory*, vol. 44, pp. 2325–2384, Oct. 1998.

[4] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. Jrnl. of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] H. Lejsek, F. H. Ásmundsson, B. Jónsson, and L. Amsaleg, "Scalability of local image descriptors: A comparative study," in *ACM Conf. on Multimedia*, 2006.

[6] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Symp. on Computational Geometry*, 2004, pp. 253–262.

[7] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, Mar 2006, ch. 3.

[8] Y. Ke, R. Sukthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in *ACM Conf. on Multimedia*, 2004, pp. 869–876.

[9] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert, "Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1111 – 1126, July 2006.

[10] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for near neighbor problem in high dimensions," in *Symp. on the Foundations of Computer Science*, 2006, pp. 459–468.

[11] J. Conway and N. Sloane, "Voronoi regions of lattices, second moments of polytopes, and quantization," *IEEE Trans. on Information Theory*, vol. 28, no. 2, pp. 211–226, 1982.

[12] ——, "Fast quantizing and decoding algorithms for lattice quantizers and codes," *IEEE Trans. on Information Theory*, vol. 28, no. 2, pp. 227–232, 1982.

[13] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. on Information Theory*, vol. 48, no. 8, pp. 2201–2214, 2002.

[14] M. Ran and J. Snyders, "Efficient decoding of the gosset, coxeter-todd and the barnes-wall lattices," in *Symp. on Information Theory*, 1998.

[15] A. Vardy and Y. Be'ery, "Maximum-likelihood decoding of the leech lattice," *IEEE Trans. on Information Theory*, vol. 39, no. 4, pp. 1435–1444, 1993.

[16] O. Amrani and Y. Be'ery, "Efficient bounded-distance decoding of the hexacode and associated decoders for the leech lattice and the golay code," *IEEE Trans. on Multimedia*, vol. 44, pp. 534–537, 1996.