FAST ALGORITHM FOR IMAGE DATABASE INDEXING BASED ON LATTICE

Mahmoud Mejdoub(1)(2), *Leonardo Fonteles*(2), *Chokri BenAmar*(1) *and Marc Antonini*(2)

(1)REGIM: Research Group on Intelligent Machines
Engineering National school of Sfax (ENIS), BP W, 3038 Sfax, Tunisia.
email: Chokri.BenAmar@enis.rnu.tn
(2)I3S laboratory
UMR 6070 University of Nice-Sophia Antipolis, CNRS, France
email: mejdoub, fonteles, am@i3s.unice.fr

ABSTRACT

Extracting feature vectors from images has been largely investigated in the literature. In contrast, browsing the image databases has not been as much studied. We propose a new efficient method for indexing a large amount of feature vectors in high dimensional space. For that purpose, we introduce a new organization of the feature vectors based on the lattice vector quantization and indexing.

1. INTRODUCTION

Along with the rapid development of multimedia devices and the internet, the amount of images has been dramatically increased in the past decade. Content-Based Image Retrieval (CBIR) [1] has mainly focused on using primitive features such as color, texture, shape, etc., for describing and comparing visual contents. But, the growth of the number of images in the database and the dimensionality of the computed feature vectors are the two major handicaps to provide efficient access to the images of the database. Therefore it is essential to use appropriate indexing techniques to search in the high dimensional feature space. Indexing techniques such as Kd-tree [2], R-trees [3], SS-trees [4] are used to realize fast search in the high dimensional feature space. These structures perform better than the scan sequential algorithm (SSA) but they are still dependent of both the feature vectors dimensions and the number of images in the database. In this paper we present a new method for speeding up the retrieval in the feature vectors space. The basic principle is as follows: feature vectors are quantized and indexed in a $\mathbb{Z}^n$ lattice. Fast retrieval is achieved by using the good properties of the lattice.

The paper is organized as follows. In section 2, we describe some indexing techniques from the literature to organize the feature vectors. In section 3, we present the adapted method to extract a fuzzy descriptor based on the combination of different wavelet basis. In Section 4 we describe the algorithm used to index a given feature vector. In section 5, we propose our browsing method to speed up the image retrieval process. In Section 6 we present some simulation results. We finally conclude in section 7.

2. RELATED WORKS

In many applications, indexing high-dimensional data has become increasingly important. In image databases, for example, the images are usually mapped to feature vectors in some high-dimensional space and queries are processed against a database of those feature vectors. The simplest method to realize similarity search is the sequential scan algorithm (SSA). Every feature vector in the database is scanned to find if it satisfies the query requirement. So, SSA depends on the number of images in the database and on the dimensions of the feature vectors. There are a lot of methods to organize the feature vectors of images in the database such that a ranked list of nearest neighbors can be retrieved without performing an exhaustive comparison with all the database image feature vectors. The state-of-the-art works done on indexing high-dimensional vectors focused mainly on two different approaches.

The first approach is based on a number of statistical algorithms such as principal component analysis [5], and latent semantic analysis [6]. These techniques are based on the observation that feature vectors in high-dimensional space are highly correlated and clustered. These methods of feature space reduction try to identify patterns in the feature space by expressing them in such a way to highlight differences and similarities that exist between feature vectors. Once these patterns are found the data can be compressed reducing the number of dimensionality. The major inconvenience of these approaches is that in many cases we will have the risk of loss of the pertinent information in the feature space resulting from the reduction of the dimensionality. The second approach is based on a number of index structures such as Kd-tree [2], R-trees [3] and SS-trees [4]. Unfortunately, currently available index structures for spatial data do not adequately support an effective indexing of more than five dimensions. The query performance of these structures degrades rapidly when the dimension of the feature vectors increases. In [7] the authors report that the query performances degrades by a factor of 12 as the dimensionality increases from 5 to 10. The major problem of the indexing structures based methods is the overlap of the bounding boxes that cover the space of the multidimensional vectors with the query window, which increases with the dimensions.

3. FEATURE VECTOR EXTRACTION

3.1 Low level feature extraction

In this section, we present the method used to extract the feature vectors. We use the wavelet transform to extract the color information. We firstly decompose the image to its individual color components RGB. The commonly used RGB color values do not correspond to human perception of color and hence are not preferred for problems like content retrieval. So we convert the images to the CIE-Lab space. Then, we use a set of orthogonal and biorthogonal discrete wavelets to transform each color component of the image in the spatio-frequency domain which is more representative than the spatial domain. We apply for that a wavelet transform with $l = 5$ levels of decomposition on each component of the CIE-Lab color space. We compute the standard deviation of the coefficients of the five low frequency bands obtained by the decomposition. Then, we extract from each decomposition level a feature vector containing a single coefficient. This greatly reduces the computational complexity of search through large databases since we obtain a $3 * l$ dimensional feature vector from each color space component. Here we obtain a 15 dimensional feature vector which reflects the low frequency information describing the color image properties.

3.2 Fuzzy descriptor extraction

In a first stage, we extract the feature vector of each image in the database using the low level visual descriptor presented in Section 3.1, and group the feature vectors in a feature matrix M. The size of this matrix is m by n where n is the number of coordinates of each feature vector and m is the number of images in the database.

We use the fuzzy c-means (FCM) [8] algorithm to partition each feature vector in the matrix M in $N_{cluster}$ (the number of clusters). Then, we compute the membership matrix U_w which contains the degree of membership of the feature vector of each image in each cluster. Degrees between 0 and 1 indicate that the feature vector has partial membership in a cluster. The matrix U_w given by (1) is of size $N_{cluster}$ by m:

$$U_w = \{U_w(j,k), 1 \leqslant j \leqslant m, 1 \leqslant k \leqslant N_{cluster}\} \quad (1)$$

where the coefficient $U_w(j,k)$ assigned to the row j and the column k of U_w indicates the degree of membership of the j^{th} image in the k^{th} cluster. The parameter $N_{cluster}$ determines the dimensionality of the feature space U_w. Since, we use in our experiments, as described later in Section 6, an image database formed by 8 categories, we set $N_{cluster}$ to 8. The advantage of the proposed approach is the reduction of the feature space dimensionality since we obtain a fuzzy feature with $N_{cluster}$ coefficients instead of 15 obtained using the low level visual descriptor.

After that, for each j^{th} image stored in the database, we extract what we call the fuzzy visual image descriptor which corresponds to the j^{th} row of U_w .

3.3 Combining different wavelet basis

We propose to use in this work three different wavelets basis (the Daubechies4, the orthogonal Beta [9, 10] and the 9-7 filters [11]). We obtain for each wavelet basis, a fuzzy visual image descriptor U_w extracted as described in section 3.2. We combine the obtained matrices $\{U_w, 1 \leqslant w \leqslant w_{number}\}$ in one fuzzy descriptor F_w given by (2); with w_{number} equal to 3, i.e., the number of wavelets basis we used.

$$F_w = [U_1, U_2, U_3] \quad (2)$$

The advantage of this approach is that the obtained coefficients that form the fuzzy visual descriptor are more meaningful and related to the image comprehension than that of the low level visual descriptor. They can then be easily combined because they have the same meaning: they reflect the degree of membership of an image into 8 categories of the image database. Besides, the proposed approach takes into consideration the relationship between images in the entire database in contrary of the low level visual descriptor that are computed only for an image independently of the other images in the database.

4. INDEXING FEATURE VECTORS

Once the feature vectors are extracted, they must be indexed in such a way that a fast and efficient retrieval in the database is guaranteed. For that purpose, we quantize each feature vector using a lattice vector quantizer (LVQ) and assign the index of the lattice vector to the corresponding feature vector. We propose to use the $\mathbb{Z}^n$ lattice in order to guarantee a fast navigation over the points in the n-dimensional space and allow a fast correspondence between a lattice vector and its index. More details are given hereinafter.

4.1 Lattice vector indexing

A lattice Λ in $\mathbb{R}^n$ is composed of all integral combination of a set of linearly independent vectors $\mathbf{a}_i$ (the basis of the lattice) such that:

$$\Lambda = \{\mathbf{x} | \mathbf{x} = u_1\mathbf{a}_1 + u_2\mathbf{a}_2 + ...u_n\mathbf{a}_n\} \quad (3)$$

where the u_i are integers. The partition of the space is hence regular and depends only on the chosen basis vectors $\mathbf{a}_i \in \mathbb{R}^m$ ($m \geqslant n$). Note that each set of basis vectors define a different lattice. Such a regular structure permits to identify the nearest vectors in the space using fast algorithms. In the case of image retrieval, since similar images have the smallest euclidian distance between their feature vectors, the use of a lattice should permit to retrieve the most similar images in an efficient way.

The LVQ is an efficient algorithm for finding the closest lattice vector $\mathbf{x}$ of a query feature vector $\mathbf{v}$. In the case of a $\mathbb{Z}^n$ lattice, the closest lattice vector with a precision γ is given by:

$$\mathbf{x} = \left[\frac{\mathbf{v}}{\gamma}\right] \quad (4)$$

where [.] stands for the 'round' operator, and γ is a scaling factor. As we will see in Sections 5 and 6 the choice of the scaling factor must be such that a good compromise between efficiency and computational complexity (speed) is achieved.

Once the feature vectors are quantized into lattice vectors $\mathbf{x}$, we may attribute a unique and decodable index for each $\mathbf{x}$. Feature vectors are obtained by the FCM algorithm and thus their coordinates are always positive numbers located in the first octant (as well as their quantized coordinates in the lattice). The proposed indexing method computes an index by classifying the lattice vectors according to their norm and the geometrical properties of the lattice. Actually, taking into account the properties of the feature vectors, an index is composed by a set of indices: an index for the *norm* (I_N), an index for the *leader* ($I_{\mathscr{L}}$), and finally an index for the *permutation* (I_p). Let us detail each of these indices in the following:

- The index for the norm (I_N) is given by the l_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} x_i$ (since the coordinates are positive numbers) of the lattice vector $\mathbf{x}$. It classifies the different lattice vectors $\mathbf{x}$ in different hyper-pyramids (shells);

- The lattice vectors lying on the same shell with index I_N are subdivided to a few number of vectors, called *leaders*. The leaders are vectors from which all the other lattice vectors of the corresponding shell can be generated by permutations and sign changes of its coordinates (here, there is no sign changes since all the coordinates are positives). The Section 4.2 explains in details how the leader indices $I_{\mathscr{L}}$ are computed;

- The index for the permutation (I_P) is computed by the Schalkwijk algorithm as in [12].

This indexing method creates an hierarchical tree-structure of indices adapted to the database indexing framework (see Section 6).

4.2 Proposed leader indexing

The proposed algorithm classifies all the leader indices in such a way that the indexing is no longer based on a greedy search algorithm or direct addressing, but on low-cost enumeration algorithm which just depends on the *quantity of leaders* instead of on the explicit knowledge of all of them.

A hyper-pyramid of radius r and dimension n is composed by all the vectors $\mathbf{v}$ such that $\|\mathbf{v}\|_1 = r$. As said before, leaders are the elementary vectors of a hyper-surface from which operations of permutations and sign changes lead to all the other vectors lying on this hyper-surface. Indeed, the leaders are vectors with positive coordinates sorted in increasing (or decreasing) order. Therefore, leaders for l_1 norm are vectors which verify the conditions below:

1. $\sum_{i=1}^{n} v_i = r$;
2. $0 \leqslant v_i \leqslant v_j$, for all $i < j$.

In the case of a l_1 norm, one can note that those conditions are linked to the *theory of partitions* in number theory [13]. Indeed, in number theory a partition of a positive integer r is a way of writing r as a sum of d positive integers (also called *part*). The number of partitions of r is given by the partition function $p(r)$ such that:

$$\sum_{r=0}^{\infty} p(r)y^r = \prod_{d=1}^{\infty} \left(\frac{1}{1-y^d}\right), \quad (5)$$

which corresponds to the reciprocal of the Euler's function [13]. Further mathematical development lead to representations of the $p(r)$ function that allow faster computation. Interested readers should refer to [13].

However, we are usually interested in shells of l_1 norm equals to r in a d-dimensional lattice with $r \neq d$. In this case, one can use the function $q(r,d)$ [13] which computes the number of partitions of r with at most d parts (in partition theory it is equivalent to the number of partitions of r with no element greater than d with any number of parts). Then, for a hyper-pyramid of norm $r = 5$ and dimension $d = 3$, we have $q(5,3) = 5$, i.e., five leaders given by: $(0,0,5)$, $(0,1,4)$, $(0,2,3)$, $(1,1,3)$,and $(1,2,2)$.

The function $q(r,d)$ can be computed from the recurrence relation[1] [13]:

$$q(r,d) = q(r,d-1) + q(r-d,d), \tag{6}$$

with $q(r,d) = p(r)$ for $d \geqslant r$, $q(1,d) = 1$ and $q(r,0) = 0$.

4.2.1 Using function $q(r,d)$ to index the leaders

As we will see in the following, Equation (6) not only gives the total number of leaders lying on a given hyper-pyramid but can also be used to provide unique indices for these leaders. To illustrate the principle of the proposed algorithm, let us suppose that the leaders of a given hyper-pyramid have been classified in a lexicographical order as:

Index value	Leader
0	$(0,...,0,0,r_n)$
1	$(0,...,0,1,r_n-1)$
2	$(0,...,0,2,r_n-2)$
3	$(0,...,1,1,r_n-2)$
$\vdots$	$\vdots$

In this way, the index of a leader $\mathscr{L}$ corresponds to the number of leaders that appear before it. For example, the leader $(0,...,1,1,r_n-2)$ should be assigned to index 3.

Consider a leader $\mathscr{L} = (x_1, x_2, ..., x_{n-1}, x_n)$ of dimension n and norm $r_n = \sum_{i=1}^{n} x_i$. Since the leaders are sorted in a lexicographical order, all the leaders with the largest coordinate g_n verifying $x_n + 1 \leqslant g_n \leqslant r_n$ appear before $\mathscr{L}$. The number of leaders with largest coordinate equal to $x_n + t$ $(t \geqslant 1)$ and norm r_n can be easily calculated using the function q of Equation (6) and is given by $q(r_n - (x_n + t), n-1)$. Clearly, computing the number of leaders with the largest coordinate equal to $x_n + t$, with norm $r = r_n$ and dimension $d = n$,is equivalent to calculate the number of leaders of norm $r_n - (x_n + t)$ with dimension $n-1$.

By introducing the function $\overline{q}(r,d,k)$ given in Appendix, which counts all the partitions of a number r with at most d parts not greater than k, we can show that the index of a leader can be computed using the following formula:

$$I_{\mathscr{L}} = \sum_{\substack{j=0 \\ \text{while } x_{n-(j+1)} \neq 0}}^{n-2} \; \sum_{i=x_{n-j}+1}^{\min[x_{n-(j-1)}, r_{n-j}]} \overline{q}(r_{n-j} - i, n-(j+1), i), \tag{7}$$

with $x_{n+1} = +\infty$ and $q(0,d) = \overline{q}(0,d,k) = 1$. Note that, when $r_{n-j} - i$ is less than or equal to i, $\overline{q}(r_{n-j} - i, n-(j+1), i) = q(r_{n-j} - i, n-(j+1))$, because in that case all vectors counted by $q(r,n)$ are leaders.

5. PROPOSED METHOD

5.1 Principle of the method

Lattice vector quantization based on $\mathbb{Z}^n$ divide the data space into hypercubes. The centroid of each hypercube is a lattice point. Each feature vector of the feature space F_w obtained as explained in Section 3.3 is quantized in a lattice point. The query procedure is given as follows:

[1] There also exist closed forms for $q(r,d)$ for the first few values of d.

1. The query feature vector is computed;
2. The query feature vector is quantized in a lattice point;
3. We exploit the good properties of the lattice space to determine the nearest lattice points of the quantized query feature vector;
4. We collect the images of the database quantized by the nearest lattice points;
5. We perform SSA on the feature vectors of the collected images.

All these steps are described in the following sections.

5.2 Building the search tree

The obtained feature space F_w computed as explained in Section 3.3 is quantized using a scaling factor γ:

$$G_w = \left[\frac{F_w}{\gamma} \right] \tag{8}$$

where [.] stands for the 'round' operator. Then, each quantized feature vector is indexed with the indices of norm, leader and permutation as explained in Section 4. The search tree is built as presented in Figure 1. It has four levels. In the first three levels, each level contains a certain number of hash tables. Each hash table has a given number of buckets, each bucket contains one key and one node. In the fourth level, each leaf node contains a list of images indexed in the same lattice point. In the root of the tree, we construct the hash table H_{root} associated to the index of norm I_N. If two images have the same I_N they are associated to the same bucket in H_{root}. The key in each bucket of H_{root} is the index of norm of the quantized feature vectors of the images existing in the database. The child of the node that exists in the bucket of H_{root} containing the key I_N, is the hash table $Child_{I_N}$. The keys of each bucket of $Child_{I_N}$ are the indices of leaders $I_{\mathscr{L}}$ of the quantized feature vectors of the images existing in the database that have the same index of norm I_N. The child of the node that exists in the bucket of $Child_{I_N}$ containing the key $I_{\mathscr{L}}$, is the hash table $Child_{(I_N, I_{\mathscr{L}})}$. The keys of $Child_{(I_N, I_{\mathscr{L}})}$ are the indices of permutation I_P of the quantized feature vectors of the images existing in the database that have the same I_N and the same $I_{\mathscr{L}}$. The child of the node that exists in the bucket of $Child_{(I_N, I_{\mathscr{L}})}$ containing the key I_P, is a leaf node $Leaf_{(I_N, I_{\mathscr{L}}, I_P)}$ containing the list of images existing in the database that have the same indices of norm, leader and permutation corresponding respectively to I_N, $I_{\mathscr{L}}$ and I_P. For example, in Figure 1, the keys of the Hash table $Child_9$ are the indices of leader of images in the database having the same index of norm 9. The keys of $Child_{(9,5)}$ are the indices of permutation of images in the database having the same indices of norm and leader corresponding respectively to (9,5). The images located in the leaf node $Leaf_{(9,5,6)}$ are image1 and image3.

image	indices(norm,leader,permutation)
image1	(9,5,6)
image2	(10,1,2)
image3	(9,5,6)
image4	(9,1,3)
image5	(9,1,2)

Table 1: The indices of norm, leader and permutation of each images presented in the search tree of Figure 1.

5.3 Retrieving the k nearest images

We implemented an image retrieval system with the Java language using Apache Tomcat server. The system allows the user to bring the query image in existing pictures from the hard disk, the web or drawings constructed on any other drawing tools.

Let us first define the nearest neighbor lattice vectors $\mathbf{x}$ of the

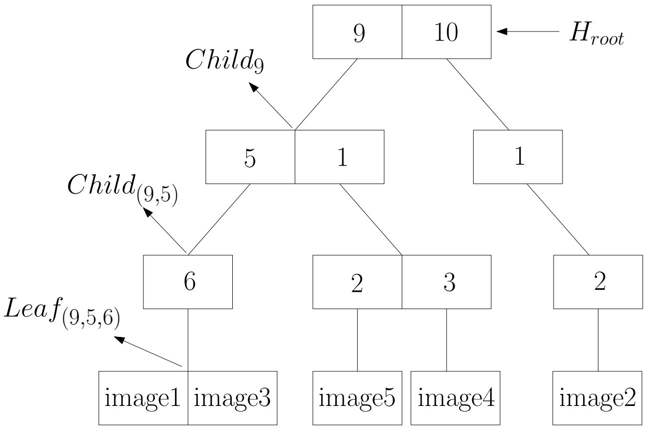

Figure 1: Example of the tree for the images given in Table 1.

origin lattice vector (the null vector $\mathbf{0}$) as[2]:

$$mask_p = \left\{ \mathbf{x} \in \Lambda, \sum_i x_i^2 = p, p \in \mathscr{P} \subset \mathbb{N} \right\} \tag{9}$$

which defines the lattice vectors at the square distance p from the origin. For example, as showed in Figure 2 for a lattice $\mathbb{Z}^2$, the neighbors at a square distance $p = 4$ from the origin of the lattice are given by $mask_4 = \left\{ \binom{0}{-2}, \binom{-2}{0}, \binom{0}{2}, \binom{2}{0} \right\}$. They correspond to the third neighborhood of the null vector $\mathbf{0}$.

The proposed basic procedure to retrieve the k nearest images of query image is given by the following steps:

1. A combined fuzzy color feature vector is extracted from the query image and quantized by the lattice vector v_{query};
2. Retrieve in the database all the images quantized by the lattice vector v_{query} as described in Section 5.4;
3. If the number k_1 of images in the database quantized by v_{query} is at least equal to k, then go to step 6 else go to step 4;
4. The set E of lattice vectors corresponding to the neighbors of v_{query} up to a maximum square distance P is defined by:
$$E = [E_1, E_2, ..., E_p, ..., E_P] \quad \text{for } p \in \mathscr{P} \tag{10}$$
with,
$$E_p = \{\mathbf{v} \in \Lambda, v = v_{query} + \mathbf{x}, \mathbf{x} \in mask_p\}; \tag{11}$$
5. Retrieve the k_2 database images quantized by vectors of E (satisfying $k_1 + k_2 \geq k$). This step is described in Section 5.4;
6. Performs the SSA on the k_1 or $k_1 + k_2$ database images and return the k nearest images to the query vector;
7. Exit.

5.4 Retrieving the database images

To retrieve the images quantized by the same lattice vector v, we proceed as follows. We compute the index of norm I_N of the quantized feature vector v. We search I_N in the root hash table H_{root}. If it does not exist, we exit the search process without returning any images. If I_N exists, we compute the leader index $I_{\mathscr{L}}$ of v and we search $I_{\mathscr{L}}$ in $Child_{I_N}$. If $I_{\mathscr{L}}$ does not exist, we exit the search process without returning any images. If $I_{\mathscr{L}}$ exists, we compute the permutation index I_P of v and we search I_P in $Child_{(I_N, I_{\mathscr{L}})}$. If it does not exist, we exit the search process without returning any images. If I_P exists, we take from the leaf node $Child_{(I_N, I_{\mathscr{L}}, I_P)}$ the images indexed by the same indices of norm, leader and permutation (I_N,$I_{\mathscr{L}}$,I_P) as the quantized feature vector v.

[2] This operation is done off-line.

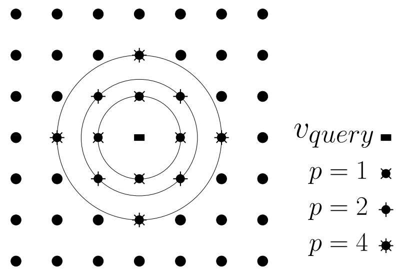

Figure 2: The lattice points located at the neighborhood of v_{query} in $\mathbb{Z}^2$.

5.5 Advantages of the proposed method

One advantage of the proposed retrieval method is that it is adapted to image databases that have an important number of images. Indeed, if we increase the number of images in the database, we obtain a lattice space more dense, consequently more useful for indexing. Besides, to search the images quantized in a given lattice point located at the neighborhood of v_{query}, we search at most only in three levels of the tree independently of the dimension of the feature space. Indeed, we parse the all three levels only if there are images in the database that are quantized in this given lattice point. Another advantage is that the speed up can be improved selecting the optimal scaling factor. If it is small then the source is expanded and the lattice will be less dense. Then, for retrieving a certain number of images the retrieval engine has to parse an important number of nearest lattice points and the speed up cost will be more important. If the scaling factor is great the source is contracted, the lattice will be very dense so the retrieval window increases but the speed up is degraded because we must perform SSA on a large number of images. So we have a trade off between the number of images in which the SSA is applied and the number of the browsed lattice points located at the neighborhood of v_{query}. To realize this trade off our solution consists on varying the scaling factor and selecting experimentally the optimal scaling factor.

6. SIMULATION RESULTS AND DISCUSSION

We conducted our tests on a subset of the COREL [14] database, formed by 8 image categories (Horses, Flowers, Buildings, Buses, Dinosaurs, Elephants, Mountains and glaciers, Lights) each containing 100 images. The images are translated scaled and rotate to obtain an image database containing 40000 images. Experiments were performed on a personal computer with configurations: Intel Pentium M 725 (1.6 GHZ), 512 MB memory DDR 333 MHz, 80 GB HDD. We have tested the performance of our proposed image retrieval method taking into account both the classical precision measures (the ratio between the number of images returned belonging to the same category as the query and the number of images required by the user) and the speed-up of the retrieval process. The dimension of the feature vectors that we used in our experiments is equal to 24, since we have combined three feature vectors each one extracted from a different wavelet basis (Daubechies4, Beta and 9_7 filters). We have used three different scaling factors ($1/2$, $1/3$ and $1/4$) and three different ranks (8, 60 and 200). The rank is the number of images required by the user. We define the speed-up parameter as:

$$\text{speed-up} = t(SSA)/t(indexing) \tag{12}$$

where $t(SSA)$ is the elapsed time for SSA method and $t(indexing)$ is elapsed time for the Kd-tree or the proposed retrieval method. Note that all feature vectors as well as the built trees of the proposed method and Kd-tree should be resident in the main memory when evaluating $t(SSA)$ and $t(indexing)$. So the speed-up is mainly dependant on computational complexity, i.e. CPU cost. We have found experimentally that for the scaling factors ($1/2$, $1/3$ and $1/4$), we obtain best performances than the Kd-tree in terms of time spent on the retrieval process for all used k (number of nearest neighbors to be retrieved) (see Tables 2 and 3). We present in Table 2 a comparison between the retrieval speed-up of the retrieval method based on lattice using the scaling factor $1/3$ and the kd-tree.

k	Kd-tree	Lattice
8	18,64	96,996
60	12,1589	66,03
200	8,375	29,538

Table 2: Speed-up comparison between the proposed indexing method and the Kd-tree.

As described in the Section 5.3, the proposed method contains two major parts. The first consists on browsing the lattice points located at the neighborhood of v_{query}. The second consists in applying the SSA on the images returned by the first part. Let us call $nb(nearest)$ the number of the browsed lattice points located at the neighborhood of v_{query} and $nb(SSA)$ the number of images in which we perform the SSA. The experiments that we made show that the best performances of our retrieval method are obtained for the scaling factor $1/3$. We can explain this, as shown in Table 3, by the fact that the increasing of $nb(SSA)$ is more responsible of the growth of the time spent on the retrieval process, than the increasing of $nb(nearest)$, i.e. , the proposed retrieval method consumes more time in its second part than in its first part. So, for the scaling factor $1/3$, we obtain the best trade-off between the number of images $nb(SSA)$ and $nb(nearest)$.

k	γ	$nb(nearest)$	$nb(SSA)$	Speed-up Lattice
8	1/2	1.4086	647.5952	43,3495
	1/3	4.167	297.6662	96,996
	1/4	27.0292	172.2377	94,3
60	1/2	7.5092	687.8592	40,9647
	1/3	112.4876	340.4619	66,03
	1/4	658.8768	220.3109	32,736
200	1/2	60.7492	820.6161	27,2889
	1/3	891.182	465.7423	29,538
	1/4	3329.5112	329.5532	8,7658

Table 3: The trade-off between $nb(SSA)$ and $nb(nearest)$ for the scaling factors γ ($1/2$, $1/3$ and $1/4$), and the respective speed-up.

For the scaling factor $1/3$, we compared also the precision of the retrieval between our indexing method and the Kd-tree.We observe in Table 4 that the results given by our methods outperform those given by the Kd-tree.

k	8	30	60	100
Kd-tree	0,938	0,84063	0,81021	0,75938
Lattice	0,938	0,8669	0,82063	0,7728

Table 4: Comparison of the precision between the proposed indexing method and the Kd-tree.

7. CONCLUSION

In this paper, we have proposed a new method for indexing a large amounts of feature vectors exploiting the good properties of the algebraic lattice $\mathbb{Z}^n$. The proposed method performs better than the Kd-tree in both the speed-up and the precision of the retrieval process. Future works will be oriented to extract feature vectors from the most pertinent object in the images.

Appendix

Given the number r, the number of parts d and the largest part k, $\overline{q}(r,d,k)$ is computed by the following algorithm:

$q = \mathbf{0}_{r+1,k+1}$; // Null Matrix of size$(r+1)\times(k+1)$
$q(0,0) = 1$;
For i from 0 *to d do*
  For j from 1 *to k do*
    For z from r to i by -1 *do*
      If $z \geqslant j$
        then
        $q(z,j) = q(z,j-1) + q(z-j,j)$;
        else
        $q(z,j) = q(z,j-1)$;
      EndIf
    EndFor
  EndFor
EndFor
$\overline{q}(r,d,k) = q(r,k)$;
return $\overline{q}(r,d,k)$;

REFERENCES

[1] J. Smith and S. Chang, "Visualseek: A fully automated content-based image query system," *Proceedings of ACM Multimedia'96*, pp. 87–98, November 1996.

[2] J. Bentley, "Multidimensional binary search trees in database applications," *IEEE Transactions on Software Engineering*, pp. 333–340, December 1979.

[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Conference*, pp. 47–57, December 1984.

[4] D. White and R. Jain, "Similarity indexing with the ss-tree," *Proceedings of the 12th International Conference on Data Engineering*, pp. 516–523, February 1996.

[5] L. Tran and R. Lenz, "Pca-based representation of color distributions for color-based image retrieval," *International Conference in Image Processing (ICIP'01)*, pp. 697–700, October 2001.

[6] S. Deerwester, S. Dumais, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, October 1990.

[7] X. Xiangyang, L. Hangzai, and W. Lide, "Index point data using algebraic lattice," *Proceedings of SPIE, Storage and Retrieval for Media Databases*, vol. 3972, no. 22, pp. 271–281, December 1999.

[8] X. Xie and G. Beni, "A validity measure for fuzzy clustering," *EEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 841–847.

[9] W. Bellil, C. B. Amar, and M. Alimi, "Beta wavelet based image compression," *International Conference on Signal, System and Design, SSD03*, vol. 1, pp. 77–82, 2003.

[10] C. B. Amar, M. Zaied, and M. A. Alimi, "Beta wavelets. synthesis and application to lossy image compression," *Advances in Engineering Software, Elsevier edition*, vol. 36, no. 7, pp. 459–474, july 2005.

[11] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 1, no. 22, pp. 205–220, april 1992.

[12] J. Moureaux, P. Loyer, and M. Antonini, "Low complexity indexing method for Z^n and D_n lattice quantizers," *IEEE Trans. Commun.*, vol. 46, no. 12, pp. 1602–1609, december 1998.

[13] G. E. Andrews, "The theory of partitions," Cambridge University Press; Reprint edition (July 28, 1998), 1998.

[14] J. Wang, J. Li, and G. Wiederhold, "Simplicity: Semantics-sensitive integrated matching for picture libraries," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 947–963, 2001.