ACQ: An Automatic Clustering and Querying Approach for Large Image Databases *

Dantong Yu and Aidong Zhang
Department of Computer Science & Engineering
State University of New York at Buffalo
Buffalo, NY 14260

Abstract

Large image collections such as web-based image databases are being built in various locations. Because of the diversity of such image data collections, clustering images becomes an important and non-trivial problem. Such clustering tries to find the densely populated regions in the feature space to be used for efficient image retrieval. In this paper, we present an automatic clustering and querying (ACQ) approach for large image databases. Our approach can efficiently detect clusters of arbitrary shape. It does not require the number of clusters to be known a priori and is insensitive to the noise (outliers) and the order of input data. Based on this clustering approach, efficient image querying is supported. Experiments demonstrate the effectiveness and efficiency of the approach.

1 Introduction

With the advent of the World Wide Web (WWW), large volume of image data are now made available over the web for information sharing and it is possible to access such large image repositories distributed throughout the world. As large web-based image databases are being built, organizing the image databases for efficient retrieval is a non-trivial problem. For supporting image retrieval, feature vectors have been extracted from image databases.

Feature-based indexing and retrieval have been commonly used in existing systems [6, 12]. However, images usually have high dimensional feature vectors, which are difficult to index. One popularly used index structure is R-tree and its variants [4, 2]. Such indexing approaches support nearest neighbor searches efficiently. One severe disadvantage of R-tree and its variants is that the bounding rectangles associated with different nodes may overlap. Overlapping becomes more serious when dimensionality increases. When dimension is more than 100 which is common in color and texture feature vectors, the advantage of the indexing will be diminished because of overlapping. When searching a R-tree, we may have to follow many paths down the tree.

Clustering has been studied to provide better organization of the images for effective retrieval [9, 10]. In general, there are two types of clustering, supervised vs unsupervised clustering. Various supervised approaches have been proposed based on the training data. However, since the web-based image databases contain a variety of images, it is usually very difficult to know the number of clusters in advance and to select training data for clustering. Thus, automated clustering of feature vectors in large image databases is needed. [1, 16, 7] provided some generic clustering algorithms. However, existing clustering algorithms are not designed to deal with the feature vectors in high-dimensional space (dimension = 64, 128, or 256). Due to the *Curse of Dimensionality* [3] (Exponential dependency of measures on the dimension), the usefulness of

* This research is supported by an NSF CAREER grant IIS-9733730.

these approaches degrade rapidly when the dimensionality increases. Furthermore, because of the diversity of the image data in such repositories, existing approaches can not be directly used to support the effective and efficient retrieval of the images in image databases.

In this paper, we present an automatic (unsupervised) clustering and querying approach (termed ACQ) for large image databases. ACQ does not require the number of clusters to be known a priori and is insensitive to the noise (outliers) and the order of input data. By intelligently applying wavelet transforms on the feature space, our clustering approach can effectively and efficiently detect clusters of arbitrary shape of high dimensional feature vectors. Based on the clustering results, efficient querying of the image database is supported. Given a query image, the relevant cluster can be quickly located. Once the query is narrowed to a specific cluster, image retrieval can proceed efficiently. Experiments demonstrate the effectiveness and efficiency of the approach.

The rest of the paper is organized as follows. Section 2 introduces the intuition of our clustering approach and Section 3 presents the details of the approach. Section 4 discusses the efficient querying. Section 5 presents the experimental results, and Section 6 gives the conclusions.

2 Using Wavelet on Feature Space

Wavelet transform is a signal processing technique that decomposes a signal into different frequency subbands (for example, high frequency subband and low frequency subband). A one-dimensional signal s can be filtered by convolving the filter coefficients α_k with the signal values:

$$s'_i = \sum_{k=0}^{M-1} \alpha_k s_{i+k-\frac{M}{2}}, \qquad (1)$$

where M is the number of coefficients in the filter and s'_i is the result of convolution. Wavelet transform provides us with a set of interesting filters. Figure 1 shows the Cohen-Daubechies-Feauveau(2,2) biorthogonal wavelet [13].

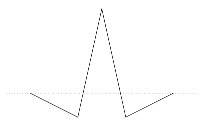


Figure 1: Cohen-Daubechies-Feauveau (2,2) biorthogonal wavelet.

To have multi-resolution representation of signals we can use discrete wavelet transform [5]. We can compute a coarser approximation of the one-dimensional input signal S_0 by convolving it with the low pass filter $\tilde{H}$ and down sampling the signal by two [5]. All the discrete approximations S_j, $1 < j < J$ (J is the maximum possible scale), can thus be computed from S_0 by repeating this process. Figure 2 illustrates the method.

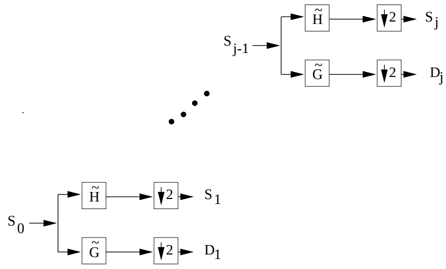
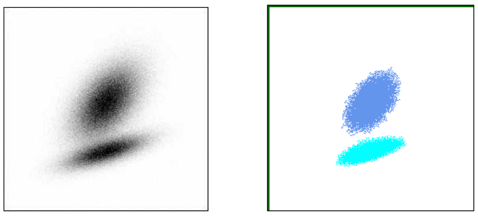


Figure 2: Block diagram of multi-resolution wavelet transform.

D_j denotes the difference between S_j and S_{j-1} and is called *detail signal* at the scale j. We can compute the detail signal D_j by convolving S_{j-1} with the high pass filter $\tilde{G}$ and returning every other sample of output. The wavelet representation of a discrete signal S_0 can therefore be computed by successively decomposing S_j into S_{j+1} and D_{j+1} for $0 \leq j < J$. This representation provides information about signal approximation and detail signals at different scales. We can easily generalize the wavelet model to d-dimensional data space in which one-dimensional transform can be applied multiple times[15].

We now discuss the relationship between multidimensional data and multidimensional signals and show how to use wavelet transform to detect the inherent relationships in the data. We propose to look at the multidimensional data space from a signal processing perspective. The collection of data in the multidimensional data space composes a d-dimensional signal. The high frequency parts of the signal correspond to the regions of the data space where there is a rapid change in the distribution of data, they are the boundaries of clusters. The low frequency parts of the d-dimensional signal which have high amplitude correspond to the areas of the data space where the data are concentrated. For example, in a two-dimensional data space, each row or column can be considered as a one-dimensional signal, so the whole data space will be a two-dimensional signal. Boundaries and edges of the clusters constitute the high frequency parts of this 2-dimensional signal, whereas the clusters themselves, correspond to the parts of the signal which have low frequency with high amplitude. When the number of data is high, we can apply signal processing techniques to find the high frequency and low frequency parts of d-dimensional signal representing the data, resulting in detecting the clusters. The key idea is to apply signal processing methods to transform the space and find the dense regions in the transformed space.

Figure 3 shows an example of a data space before and after transform using the *hat-shape* filter shown in Figure 1. Such filters emphasize regions where points cluster, but simultaneously tend to suppress weaker information in their boundary. This means clusters in the data automatically stand out and clear regions around them, so that they become more distinct. As a side-effect, applying wavelet transform can automatically remove noise data that do not belong to any of the clusters. This dataset contains 500,000 data in the two clusters plus 25,000 randomly distributed noise data. As the figure shows, the clusters in the transformed space are more salient and thus easier to be found.



a) b)

Figure 3: a)Original space; b)Transformed space

3 Clustering in High Dimensional Feature Space

In the high dimensional feature space, the actual dense areas are very small compared to the whole space. A hash table can be an effective method to exclusively deal with this kind of situation. In this section, we present a novel clustering algorithm, termed ACQ, and provide an indepth analysis of the data structure used, the problems associated in adapting this type of data structure and how we solve them.

Given a set of feature vectors $\mathbf{v_i}, 1 \leq i \leq N$, the goal of the algorithm is to detect clusters and assign labels to the vectors based on the cluster they belong to. We use wavelet transform as a tool for facilitating clustering. The main idea here is to efficiently represent high dimensional data in limited memory and perform wavelet transform as well as connected component analysis on this representation. The outline of the algorithm is given below:

1. Quantize feature space, aggregate feature vectors into the cells and construct a hash table H.
2. Apply wavelet transform on H and save the transformed data in a new hash table H'.
3. Find the connected components (clusters) in H'.
4. Assign labels to the cells and make the lookup table.

3.1 Feature Space Representation

Since we use discrete wavelet transform, before applying the transform, the feature space should be quantized. In quantization, each dimension A_i in the d-dimensional feature space is divided into m_i intervals. If we assume that m_i is equal to m for all the dimensions, there would be m^d cells in the feature space. Then the corresponding cell for the feature vectors will be determined based on their attribute values. For each cell we count the number of feature vectors contained in it and use this count as the data of the cell (termed *count value*). In the quantized space, every cell $\mathbf{c_i}$ is represented by $\langle c_{i1}, c_{i2}, \ldots, c_{id} \rangle$ which is called the *key* or *index* for $\mathbf{c_i}$, where

$c_{ij} = [l_{ij}, h_{ij})$ is the right open interval in the partitioning of dimension A_j. The address of a cell in the hash table can be calculated by applying appropriate hash function on the index of a cell. A hash table requires much less storage than a multidimensional matrix, which was used in [11]. Specifically, the storage requirement can be reduced from $O(m^d)$ to $\Theta(N' \times d)$, where N' is the number of nonempty cells in the quantized feature space.

With hashing, a cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$ with its count value is stored in the hash bucket $h(\mathbf{c_i})$; that is, a **hash function** h is used to compute the address for the cell $\mathbf{c_i}$. Formally, the hash function h maps the universe U of $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$ into the entries in the *hash table* $H[0 \ldots n-1]$, where n is the number of buckets in the hash table. That is, $h : U \to 0, 1, ..., n-1$.

Designing a hash function. In our approach, since hashing is performed frequently, the time spent on hashing directly affects efficiency. Also, both applying wavelet transform and finding connected components require the neighborhood information, that is, to locate neighbor cells of a given cell. However, hashing permits any element to be mapped into any of the hash table buckets. Thus, it introduces the problem of determining or locating the neighbors of a cell. Another issue is the collision problem in which two or more cells may be hashed into the same bucket. Our goal to design the hash function is to achieve efficiency, easy computation of neighbor cells, and minimal collision.

We now define our hash function. We randomly generated an integer matrix $A_{l\times d}$, where $l = log_2 n$ and n is the number of hash table buckets as follows:

$$A_{l\times d} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,d} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,d} \\ \cdots & \cdots & \cdots & \cdots \\ r_{l,1} & r_{l,2} & \cdots & r_{l,d} \end{pmatrix}.$$

For each key $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$, hash function $h(c_{i1}, c_{i2}, \ldots, c_{id})$ is:

$$h(c_{i1}, c_{i2}, \ldots, c_{id}) =$$
$$\begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,d} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,d} \\ \cdots & \cdots & \cdots & \cdots \\ r_{l,1} & r_{l,2} & \cdots & r_{l,d} \end{pmatrix} \bigodot \begin{pmatrix} c_{i1} \\ c_{i2} \\ \cdots \\ c_{id} \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \cdots \\ z_l \end{pmatrix} \tag{2}$$

where $\bigodot$ is equivalent to matrix multiplication in binary operations. Specifically, for each cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$, we calculate:

$$z_k = \prod_{\oplus j=1}^{d} \#(r_{k,j} \wedge c_{ij}), \quad 1 \le k \le l,$$

where $\oplus$ and $\#$ are defined as:

- $\oplus$ is bitwise exclusive OR. When the operands are one bit, this operator is a bit operator. When the two operands a and b are binary vectors with the same length, the result will be a binary vector which has the same length as a and b.
- $\#$ is defined as $\#(a) = a_1 \oplus a_2 \oplus ... \oplus a_\mu$, where $a_i \in \{0, 1\}$, $1 \le i \le \mu$, is the i-th bit of a and μ is the length of binary vector a.
- Binary operator $\wedge$ is same as applying bitwise AND operation on two vectors.

Result $\mathbf{z} = \langle z_1, z_2, \ldots, z_l\rangle$ will be a string of 0 and 1, which is the address of the entry where cell $\mathbf{c_i}$ is located in the hash table.

It can be proved that the hash function given in Equation 2 maps a cell in any hash bucket with equal probability, which reduces collision. We further resolve collision by extended queuing with each bucket. Due to space limitation, we omit the details here.

3.2 Accumulative Calculation of Wavelet Transform on Hashed Feature Space

In ACQ, wavelet transform must be applied on hashed representation H of the quantized space to generate a new hash table. By scanning through the hash table H and convolving the filter given in Equation 1 with each cell and its neighbors, we generate new cells in the transformed space. For any nonempty cell $\mathbf{c_i} = \langle c_{i1}, c_{i2}, \ldots, c_{id}\rangle$, the cells which will contribute to its count value in the transformed space along dimension A_j are, $\mathbf{c_k} = \langle c_{i1}, c_{i2}, \ldots, c_{ij} + k, \ldots, c_{id} \rangle$, where $-\frac{M}{2} \leq k \leq \frac{M}{2}$. All the cells stored in the hash table will get new values after wavelet transform is applied (See Figure 4). Also, because of the convolution operation in wavelet transform, some of the previously empty cells will become nonempty by receiving contributions from their neighboring cells. We call each potential nonempty cell a *receiver* and each old nonempty cell a *contributor*. In traditional implementation of wavelet transform, each receiver knows which cells to ask for contributions. In a multidimensional array implementation, every cell is considered as a potential receiver. So the algorithm has to scan through the entire space of cells which we should try to avoid in high dimension case because of the exponential growth in the number of cells. Furthermore, in the case of hashed implementation we only have information about the cells which are nonempty, there are many potential receivers about whose hashed location we have no knowledge. Therefore, it is not possible to use traditional scanning algorithms directly on the hashed quantized feature space.

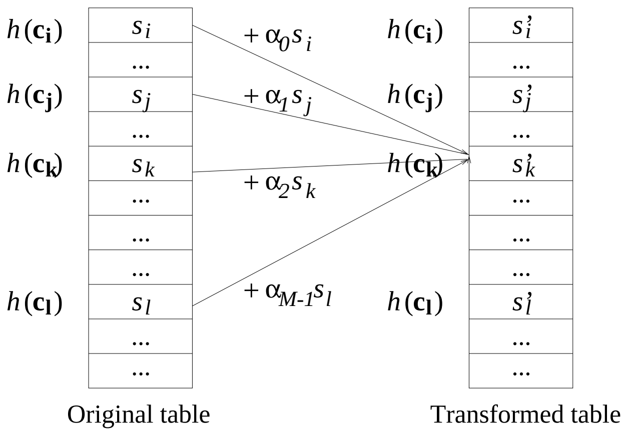


Figure 4: Traditional approach of calculating wavelet transform.

In our approach, each contributor knows its receivers and the addresses of the receivers can be calculated by using the hash function. So, instead of receivers asking for values, contributors distribute values to receivers. Thus, it is sufficient to just scan through the contributors which are already saved in the hash table. With slight modifications, we can rewrite Formula 1 as:

$$s'_{i+j} = \alpha_{\frac{M}{2}-j} s_i + \sum_{k=0}^{\frac{M}{2}-j-1} \alpha_k s_{i+j-\frac{M}{2}+k} + \sum_{k=\frac{M}{2}-j+1}^{M-1} \alpha_k s_{i+j-\frac{M}{2}+k}, \quad (3)$$

where $-M/2 \leq j < M/2$.

Using this formula while scanning the hash table H, each old nonempty cell or contributor is multiplied by a coefficient and the result is accumulated into its *receiver* cells which are hashed into the new table H' (See Figure 5).

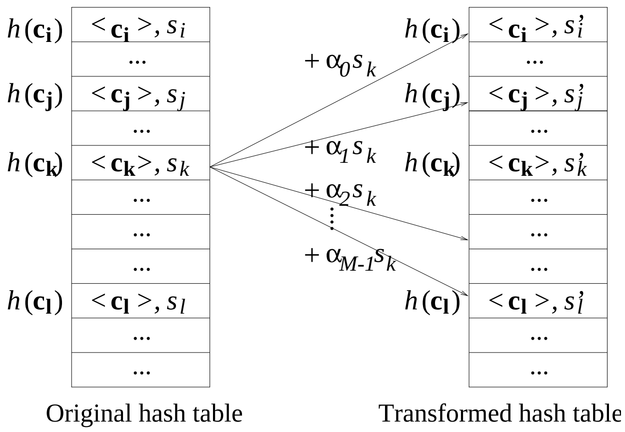


Figure 5: Accumulative approach of calculating wavelet transform.

Due to the generation of new nonempty cells, the number of cells in the new hash table H' will be increased after wavelet transform is applied. In many cases, a large number of new nonempty cells tend to have very small count values. Many of these low count values are expected to be caused by the outliers rather than the actual clusters. Also, the actual cluster shapes are distorted on the surfaces because of the directionality property of convolution operation used in wavelet transform. Removing low count value cells by applying a threshold on the count values will effectively remove majority of the outliers and help preserving the original shape of the clusters. In addition, reduction in number of cells in the hash table is expected to improve the time complexity of the algorithm. In the hash table H' constructed after applying wavelet transform on the hash table H, only the significant cells, which are in the transformed space with the count value above a threshold, are stored. The threshold used to determine the significant cells plays an important role in the quality of clustering and outlier removing. The details of determining the threshold can be found in [15].

3.3 Finding Connected Components in Hash Table

The hash table is essentially a graph $G = (V, E)$, where $V = \{\mathbf{c_i} | \ \mathbf{c_i}$ is a significant cell in transformed space$\}$ and $E = \{(\mathbf{c_1}, \mathbf{c_2}) | \ D(\mathbf{c_1}, \mathbf{c_2}) \leq \varepsilon\}$, where D is an appropriate distance function. We use $\varepsilon = 1$ and distance D as *City-block distance* or $\|l_1\|$ metric:

$$D_{\|l_1\|}(\mathbf{c_1}, \mathbf{c_2}) = \sum_{i=1}^{d} | \ c_{1i} - c_{2i} \ | \ .$$

There is an edge between two cells if and only if their indices differ on only one dimension. So every significant cell has at most $2d$ neighbors. Since finding connected components require the neighborhood information frequently, an efficient method to determine the neighbors will have positive effect on the overall running time. We present a theorem which rediscovers the neighborhood information efficiently.

Theorem 1 *Given cells* $\mathbf{c_j} = \langle c_{j_1}, c_{j_2}, \ldots, c_{j_k}, \ldots, c_{j_d} \rangle$ *and* $\mathbf{c'_j} = \langle c_{j_1}, c_{j_2}, \ldots, c'_{j_k}, \ldots, c_{j_d} \rangle$, *where* $\mathbf{c_j}$ *and* $\mathbf{c'_j}$ *are different by only k-th dimension. If we know the hashed index value of* $\mathbf{c_j}$*, then the hashed index value of* $\mathbf{c'_j}$ *can be computed using the following formula*[1]*:*

$$h(\mathbf{c'_j}) =$$

$$h(\mathbf{c_j}) \oplus \begin{pmatrix} \#(c_{j_k} \wedge r_{1,k}) \\ \#(c_{j_k} \wedge r_{2,k}) \\ \cdots \\ \#(c_{j_k} \wedge r_{d,k}) \end{pmatrix} \oplus \begin{pmatrix} \#(c'_{j_k} \wedge r_{1,k}) \\ \#(c'_{j_k} \wedge r_{2,k}) \\ \cdots \\ \#(c'_{j_k} \wedge r_{d,k}) \end{pmatrix} \quad (4)$$

Thus, given the index of a cell, the bucket number of the neighboring cells can be computed by the hash function in Equation 4. The *clusters* are the *connected components* of graph G, which can be found by a depth-first-search algorithm. ACQ starts from the first bucket in the hash table, assigns it the first cluster number and searches the cells connected to it. It continues to scan the hash table H' until all the cells are visited.

[1]See [15] for the proof.

3.4 Labeling Cells and Making Look Up Table

Applying wavelet transform on the cells in $\{\mathbf{c_j} : 1 \leq j \leq \mathcal{J}\}$ results in a new feature space and hence new cells $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$. Given the set of cells $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$, ACQ detects the connected components in the transformed feature space. Each connected component is a set of cells in $\{\mathbf{t_k} : 1 \leq k \leq \mathcal{K}\}$ and is considered as a cluster. Each cluster w, $w \in \mathcal{F}$, will have a cluster number w_n, where $\mathcal{F}$ is the set of detected clusters. In this step we label the cells in the original quantized feature space that are included in a cluster, with its cluster number:

$$\forall w \ \ \forall \mathbf{t_k}, \ \mathbf{t_k} \in w \Longrightarrow l_{\mathbf{t_k}} = w_n, \quad w \in \mathcal{F},$$

where $l_{\mathbf{t_k}}$ is the label of the cell $\mathbf{t_k}$. The clusters that are found are in the transformed space. Thus, they cannot be directly used to define the clusters in the original space. We make a lookup table LT which is also a hash table to map the cells in the transformed feature space to the cells in the quantized space. Each entry in the table specifies the relationship between a cell in the transformed space and the corresponding cell(s) in the quantized space. So the label of each cell in the original space can be easily determined. Finally the label of each cell in the quantized space is assigned to all the feature vectors contained in that cell, and thus the clusters are determined. Formally,

$$\forall \mathbf{c_j} \forall \mathbf{v}, \ \mathbf{v} \in \mathbf{c_j} \Longrightarrow l_{\mathbf{v}} = l_{\mathbf{c_j}},$$

where $l_{\mathbf{v}}$ is the cluster label of feature vector $\mathbf{v}$.

3.5 Time Complexity

It can be proved that by introducing the hash data structure to represent the dataset, we can cluster d-dimensional data in the time complexity of $O(Nd \ logn)$, where N is the number of data, d is the number of dimensions, and n is the number of buckets in the hash table, $n \leq N$. Detailed analysis can be found in [15].

4 Querying

To retrieve database images which are similar to a query image, the distance between the query image and database images will be computed. We use Euclidean distance D to calculate the distance between the feature vector of the query image and the feature vectors of database images. The images with the distance smaller than a given threshold will be retrieved. Linear search within a cluster is straightforward, but might not be efficient when cluster is large. We want to narrow down the search scope.

Given feature vectors $\mathbf{v} = (v_1, v_2, \ldots, v_d)$ and $\mathbf{e} = (e_1, e_2, \ldots, e_d)$, we denote the distance D_i on *i-th* dimension as $D_i(\mathbf{v}, \mathbf{e}) = \mid v_i - e_i \mid$. If $D_i(\mathbf{v}, \mathbf{e})$ is smaller than a given threshold ϵ, we call $\mathbf{v}, \mathbf{e}$ as *i-neighbors*. We introduce the following definitions:

Definition 1 *(Neighborhood N_ϵ^i on i-th dimension) Given a feature vector* $\mathbf{v}$ *and distance threshold ϵ, Neighborhood N_ϵ^i will be defined as: $N_\epsilon^i(\mathbf{v}) = \{\mathbf{e} | D_i(\mathbf{v}, \mathbf{e}) \leq \epsilon$, $\mathbf{v}$ and $\mathbf{e}$ are in the same cluster $\}$.*

Definition 2 *(Neighborhood N_ϵ) Given a feature vector* $\mathbf{v}$ *and distance threshold ϵ, Neighborhood N_ϵ will be defined as: $N_\epsilon(\mathbf{v}) = \{\mathbf{e} | D(\mathbf{v}, \mathbf{e}) \leq \epsilon$, $\mathbf{v}$ and $\mathbf{e}$ are in the same cluster $\}$.*

We use $\|N_\epsilon(\mathbf{v})\|$ to denote the number of vectors in set $N_\epsilon(\mathbf{v})$(*cardinality*). We can prove that:

$$N_\epsilon(\mathbf{v}) \subseteq \bigcap_{1 \leq i \leq d} N_\epsilon^i(\mathbf{v}). \tag{5}$$

Obviously, neighborhood N_ϵ^i on *i-th* dimension is much easier to locate than neighborhood N_ϵ because only one dimension is considered. For any query $\mathbf{q}$, following Formula 5, if $N_\epsilon^i(\mathbf{q})$, $1 \leq i \leq d$, can be located efficiently, we only need to pick one of them and search through it to get $N_\epsilon(\mathbf{q})$. Thus time performance can be improved. We provide an index strategy to discover the information of $N_\epsilon^i(\mathbf{q})$, to help fast retrieval.

Index on the feature vectors. One popular used index structure is R-tree and its variants [4, 2]. One disadvantage of R-tree and its variants is that the bounding rectangles associated with different nodes may overlap. Overlapping becomes more serious when dimensionality increases. When dimension is more than 100, the advantage will be diminished by overlapping(overlapping is close to 100%). When searching a R-tree, we may have to follow multiple paths down the tree. We provide an efficient way to support image retrieval.

We build d arrays to keep all the neighborhood information on every single dimension for each cluster. For *i-th* dimension, we order the feature vectors by *i-th* component and save it into *i-th* array. It takes $O(d \times n_{\mathcal{C}} \times \log(n_{\mathcal{C}}))$ to build indices, where $n_{\mathcal{C}}$ is the number of vectors in the cluster $\mathcal{C}$.

Compared with R-tree and its variants, our index structure doesn't at all get affected by overlapping even when dimension $d > 100$. The maintenance is simple and fast. Single insertion can be saved in an extended chain, it takes $O(d \times \log n_{\mathcal{C}})$. When enough insertions have been accumulated, they would be inserted into arrays within time $O(d \times n_{\mathcal{C}})$.

Retrieve algorithm. Let $(\mathbf{v})_i$ denote the *i-th* component of $\mathbf{v}$. For a given query $\mathbf{q}$, if there exists a feature vector $\mathbf{v} \in N_\epsilon^i(\mathbf{q})$, $\forall \mathbf{e} \in N_\epsilon^i(\mathbf{q})$, $(\mathbf{v})_i \leq (\mathbf{e})_i$, then we call $\mathbf{v}$ as $\mathbf{q}$'s *leftmost i-neighbor*, denoted as $\mathbf{v}_{\mathbf{q}_\mathbf{l}^\mathbf{i}}$. Similarly, we can define $\mathbf{q}$'s *rightmost i-neighbor* $\mathbf{v}_{\mathbf{q}_\mathbf{r}^\mathbf{i}}$. Clearly, if $N_\epsilon^i(\mathbf{q}) \neq \emptyset$, the members of $N_\epsilon^i(\mathbf{q})$ are between $\mathbf{q}$'s leftmost *i*-neighbor $\mathbf{v}_{\mathbf{q}_\mathbf{l}^\mathbf{i}}$ and rightmost *i*-neighbor $\mathbf{v}_{\mathbf{q}_\mathbf{r}^\mathbf{i}}$ in the *i-th* index array. (See Figure 6). By binary search on the index arrays, we can easily calculate $\mathbf{q}$'s leftmost and rightmost *i*-neighbors, where $1 \leq i \leq d$. Thus $N_\epsilon^i(\mathbf{q})$ can be obtained. We choose the $N_\epsilon^i(\mathbf{q})$ with minimum cardinality($\|N_\epsilon^i(\mathbf{q})\|$) as the searching set. By searching through $N_\epsilon^i(\mathbf{q})$, we can get the closest neighbors to $\mathbf{q}$ within the cluster (See Algorithm 1).

The running time of step 1 is $O(d \times \log n_{\mathcal{C}})$, where $n_{\mathcal{C}}$ is the number of vectors. The running time of step 2 is $O(d)$. The running time of step 3 is $O(d \times \|N_\epsilon^j(\mathbf{q})\|)$, where $N_\epsilon^j(\mathbf{q})$ has the minimum cardinality among all $N_\epsilon^i(\mathbf{q})$, $1 \leq i \leq d$. The running time is determined by $\|N_\epsilon^j(\mathbf{q})\|$. Clearly, in average, $N_\epsilon(\mathbf{q}) \simeq N_\epsilon^i(\mathbf{q})$.

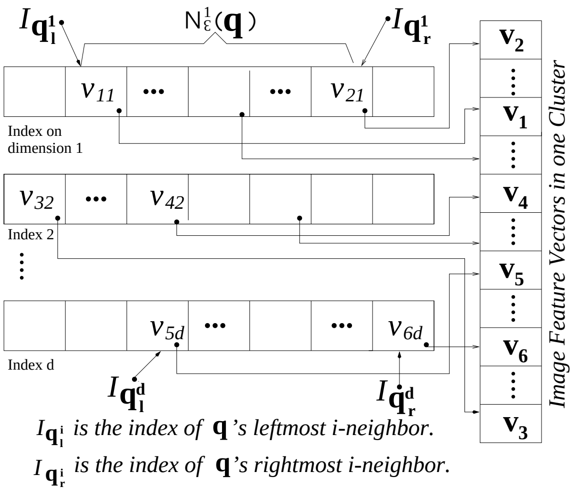


Figure 6: Index structure for feature vectors.

Algorithm 1

Input: Query image $\mathbf{q}$
Output: Images that are similar to $\mathbf{q}$

1. For each i, $1 \leq i \leq d$
   a. Binary search index array a_i for $\mathbf{q}$'s leftmost i-neighbor $\mathbf{v}_{\mathbf{q}_l^i}$ and rightmost i-neighbor $\mathbf{v}_{\mathbf{q}_r^i}$.
   b. If $\mathbf{v}_{\mathbf{q}_l^i}$ and $\mathbf{v}_{\mathbf{q}_r^i}$ don't exist then no close neighbors, output $\emptyset$ and stop.
   c. Compute $N_\epsilon^i(\mathbf{q})$
2. Get $N_\epsilon^j(\mathbf{q})$ with $minimum_{1\leq i\leq d}$ $\|N_\epsilon^i(\mathbf{q})\|$.
3. Linear search in $N_\epsilon^j(\mathbf{q})$ to get $N_\epsilon(\mathbf{q})$.

5 Experiments

In this section, we evaluate the performance of ACQ and demonstrate its effectiveness and efficiency on image data sets. Tests were done on Photodisc data sets. We present the experiments on 128-*dimension* feature vectors of color histogram of images.

5.1 Photodisc Datasets

For the experiments, we established three image databases using the images from PhotoDisc Company. DB1 contains 770 images, DB2 contains 6930 images, and DB3 contains 37730 images. We experimented our clustering approach on color feature vectors of the three image databases. We calculated the color feature vectors using the quantization method of color histograms given in [14]. All of the tests were done on a Sun UltraSPARC 168 MHz machine having 1024 MB of main memory.

5.2 Clusters in DB2

Since we performed unsupervised clustering and didn't assume any pre-knowledge about clusters, all the image clusters should be evaluated by human perception. We detected 10 clusters in DB2 under quantization $m = 24$. Figure 7 shows some sample images in six detected clusters. We also manually clustered the image database and found 12 clusters. We call them real clusters. By comparing the real clusters with the detected clusters, we observed that in our clustering, the two clusters with pink color and red color were merged into a single cluster, and the two clusters with dark brown and brown were merged into a single cluster. Both of the merged two feature sets are semantically very similar and their feature vector sets overlap. Thus, the 10 clusters give quite accurate grouping of the color image database. Due to space limitation, the clustering results on other databases are omitted.

5.3 Quantization

Quantization affects the quality of clustering. All the grid-based approaches for clustering suffer from the *Modifiable Area Cell Problem (MAUP)* addressed in [8]. The problem occurs in terms of scaling and aggregation. The problem of scaling is in selecting appropriate size and number of cells to represent the data. Aggregation is the problem of summarizing the data *contained* in each cell. In our context, scaling is an inherent problem in what a user can call a cluster, in other words, the definition of cluster. As Openshaw and Taylor stated, it seems very unlikely that there will ever be either a purely statistical or mathematical solution for MAUP [8]. To have an optimal quantization, application domain information should be incorporated. We may use a heuristic-based approach to experimentally look for a good quantization. We can start with an over-quantized space and try to find reasonable

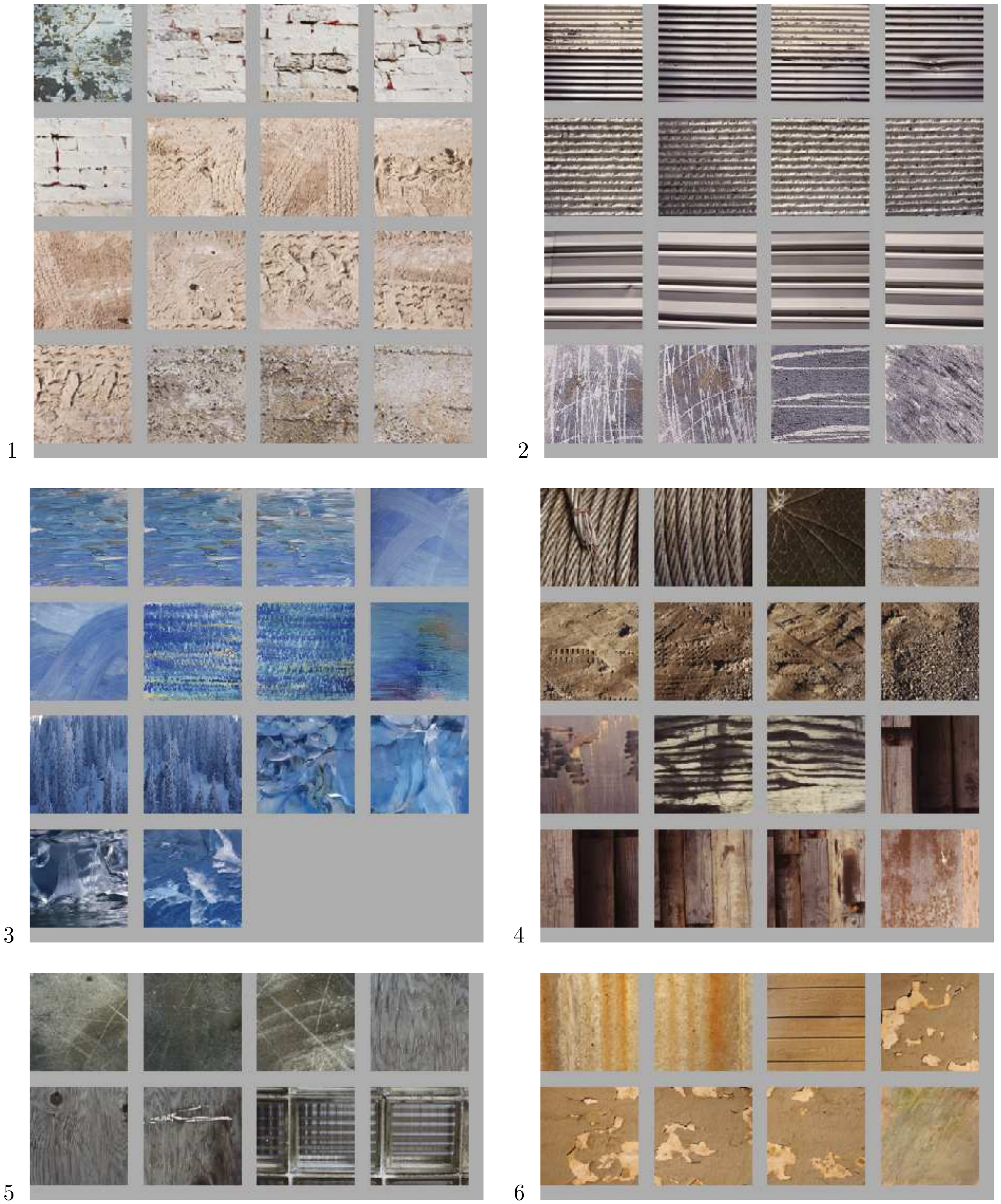


Figure 7: Sample images from 6 detected clusters in DB2.

clusters. If necessary, we then increase the size of cells and repeat the process until we get some acceptable clusters.

Table 1 shows the numbers of clusters using different quantizations m (m = 16, 20, 24, 32, 48, 64, 96, 128) for image database DB1, DB2, and DB3.

Image Databases	m							
	16	20	24	32	48	64	96	128
DB1	11	39	90	377	464	452	472	521
DB2	2	6	10	25	121	155	285	551
DB3	1	1	1	5	20	44	63	73

Table 1: Numbers of of clusters for three image databases and different quantizations m.

As shown in Table 1, when m increases, the number of clusters also increases. In general, when the quantization value m is too low (very coarse quantization), more feature vectors will be assigned to the same cell, and there is higher probability for the vectors from different clusters to belong to the same cell. This results in merging of the clusters, In contrast, if the quantization value m is too high (very fine quantization), each vector will be in a separate cell which might be far from the other cells. It can result in many very fine clusters which should have been in one cluster when m is small. Thus one big cluster might be decomposed. By choosing different quantization m, our approach can be used to cluster at different granularities. The selection of m is determined by user's requirement. This provides the flexibility to modify queries based on initial results. Figure 8 a) and b) are two clusters when m = 48, and c) is one cluster when m=32. As we noticed, a) and b) in Figure 8 are two subclusters of c).

From Table 1, we also found that many clusters were detected in DB1 when m is high. When the number of images is small and quantization m is high, the quantized space is very sparse. Thus every feature vector is isolated from each other and becomes a single cluster. This is the reason why we detected so many clusters on DB1. So our approach works better on very large image databases than small image databases. From statistic point of view, when the number of images is small, there might not have any patterns in the whole image feature space. Thus clusters may not get detected. When the number of images increases, the image feature space forms certain patterns and clusters appear more clearly. So automatic clustering can be applied to catch them.

5.4 Timing on different quantizations

Quantization m also affects running time. When m is small (very coarse quantization), the total number of cells is small. Thus it takes less time to complete clustering. When m is large, it takes longer time to complete clustering. Running time is also affected by the number of data points. Figure 9 shows CPU-time on clustering DB1, DB2 and DB3 under different quantizations.

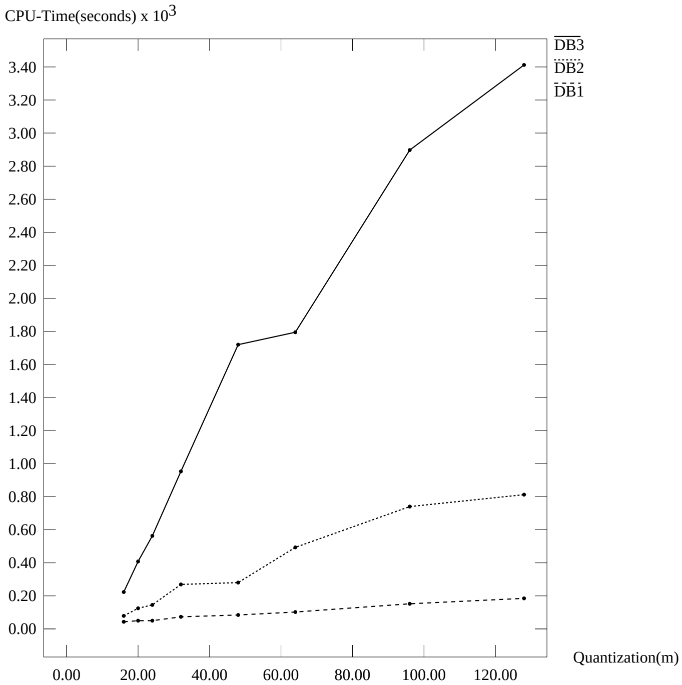
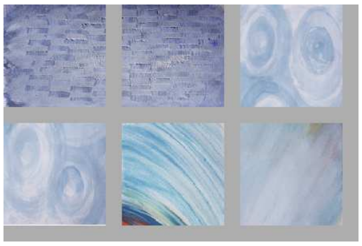


Figure 9: Run time under quantizations.

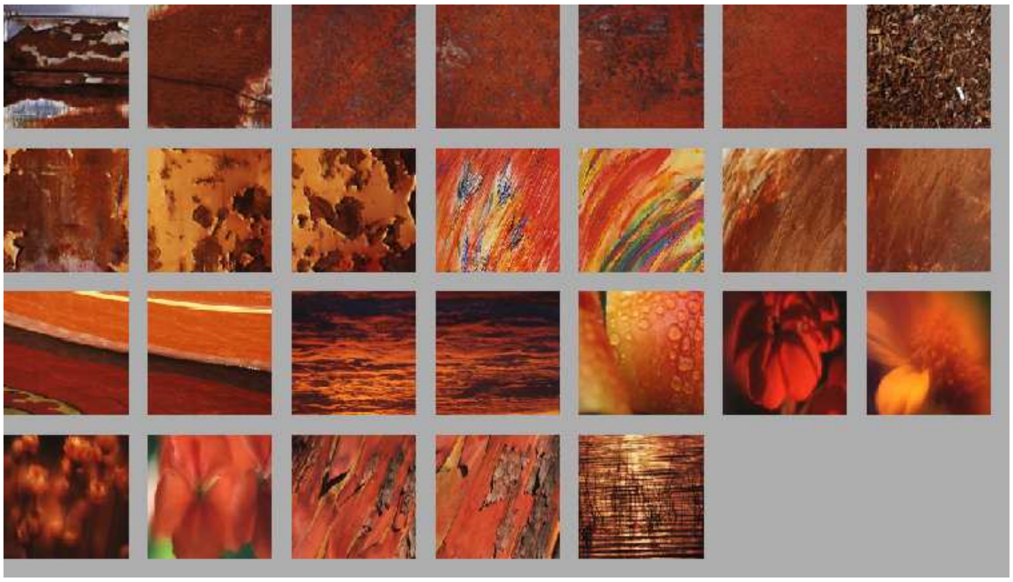
5.5 Retrieve Based on clusters

Our approach works effectively on content-based image retrieval. Figure 10 shows the query results based on clustering for a given query image.

a)  b) 

c) 

Figure 8: Finer clusters will be the subclusters of a coarser cluster



a) Query Image; b) Result;

Figure 10: Retrieve based on color

6 Conclusion

We have presented an automatic clustering and querying approach, termed ACQ, for large image databases. This approach can efficiently detect clusters of arbitrary shape without requiring the number of clusters to be known a priori and is insensitive to the noise (outliers) and the order of input data. By intelligently applying wavelet transforms on the feature space, we have shown that clustering can be done to effectively and efficiently detect clusters of arbitrary shape of high dimensional feature vectors. Based on the clustering results, efficient querying of the image database is performed by narrowing the query to a specific cluster and then conducting an efficient search of the similar images. Experiments conducted on large image data sets have demonstrated that our approach is highly effective and efficient in supporting image retrieval.

References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 94–105, Seattle, WA, 1998.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[3] S. Berchtold and D.A. Keim. High-dimensional index structures:database support for next decade's applications. In *Proceedings of ACM-SIGMOD International Conference on Management of Data, Tutorial Sessions*, pages 501–501, Seattle, June 1998.

[4] A. Guttman. R-Trees: A Dynamic Index for Geometric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[5] S. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:674–693, July 1989.

[6] B.S. Manjunath and W.Y. Ma. Texture Features for Browsing and Retrieval of Image Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[7] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th VLDB Conference*, pages 144–155, Santiago, Chile, 1994.

[8] S. Openshaw and P. Taylor. *Quantitative Geography: A British View*, chapter The Modifiable Areal Unit Problem, pages 60–69. London: Routledge, 1981.

[9] E.J. Pauwels, P. Fiddelaers, and L. Van Gool. DOG-based unsupervized clustering for CBIR. In *Proceedings of the 2nd International Conference on Visual Information Systems*, pages 13–20, San Diego, California, December 1997.

[10] G. Sheikholeslami, W. Chang, and A. Zhang. Semantic clustering and querying on heterogeneous features for visual data. In *The proceedings of the 6th ACM International Multimedia Conference (ACM MULTIMEDIA '98)*, pages 3–12, Bristol, UK, September 1998.

[11] G. Sheikholeslami, S. Chatterjee, and A. Zhang. *WaveCluster*: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the 24th VLDB conference*, pages 428–439, New York City, August 1998.

[12] J. R. Smith and S. Chang. Transform Features For Texture Classification and Discrimination in Large Image Databases. In *Proceedings of the IEEE International Conference on Image Processing*, pages 407–411, 1994.

[13] Greet Uytterhoeven, Dirk Roose, and Adhemar Bultheel. Wavelet transforms using lifting scheme. Technical Report ITA-Wavelets Report WP 1.1, Katholieke Universiteit Leuven, Department of Computer Science, Belgium, April 1997.

[14] J. Wang, W. Yang, and R. Acharya. Color Clustering Techniques for Color-Content-Based Image Retrieval. In *the Fourth IEEE International Conference on Multimedia Computing and Systems (ICMCS'97)*, pages 442–449, Ottawa, Canada, June 1997.

[15] D. Yu, S. Chatterjee, G. Sheikholeslami, and A. Zhang. Efficiently detecting arbitrary shaped clusters in very large datasets with high dimensions. Technical Report 98-8, State University of New York at Buffalo, Department of Computer Science and Engineering, November 1998.

[16] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, 1996.