Multiple Features Indexing in Image Retrieval Systems

Augusto Celentano, Stefano Sabbadin

Università Ca' Foscari di Venezia, Dipartimento di Informatica
Via Torino 155, 30172 Mestre (VE), Italia
auce@unive.it,ssabbadi@dsi.unive.it
phone +39 0412908425, fax +39 0412908419

Abstract. In this paper we discuss the problem of indexing a large collection of images where each image is described by several features. Each feature defines a multidimensional space where images are points, and the similarity between images is computed as the distance between points.
We propose to approach multiple features indexing by maintaining a separate index structure for each feature, and merging the results during query processing. This approach, provides several advantages because allows different metrics to be used or different weights to be assigned to different features, leading to better retrieval quality. Some techniques for merging the sets retrieved by considering each feature separately are discussed, and experimental results on real as well as simulated data are shown.

1 Introduction

Traditional approaches to database content modelling use alphanumeric data to represent documents. Multimedia documents, containing image, audio and video components, can be described by attaching textual descriptors (i.e., legends) to non textual and unformatted content. Indexing and retrieval is then performed on such descriptors [1]. Limiting our discussion to image databases, a query submitted to an image retrieval system would be composed by descriptors of the target image, and retrieval would take place by comparing descriptors according to some similarity criteria.

A more ideal situation would be the one in which queries refer to features computed from the image content: returned images could then be ranked according to the degree of actual content matching.

Simple systems tend to rely on a representation that is based on numerical vectors to describe features such as color, shape, texture, direction of lines, and position of objects within images: queries are "by example", i.e., an image is used as a query, whose vector is compared against those associated to the database images, and similarity is evaluated by computing the distance or the correlation between them.

In this paper we discuss the problem of indexing a large collection of images each one described by several features. Each feature defines a multidimensional space where images appear as points (*spatial databases*) [14], and the similarity between images is computed as the distance between points.

Multiple features can be indexed according to two different approaches: by merging the feature vectors before creating a global index, or by leaving them apart, creating a separate index structure for each feature, and merging the results during query processing. This latter approach, while less performing in terms of indexing and retrieval time, provides several advantages because allows different metrics to be used or different weights to be assigned to different features, leading to better retrieval quality. Several techniques can be devised for merging the sets retrieved by considering each feature separately.

This paper is organized as follows. In Section 2 we briefly overview the relevant literature on content based retrieval and multi-dimensional indexing. Section 3 describes the *VAMSplit R-Tree* indexing technique, well suited for indexing static collections, such as CD-ROMs databases. In Section 4 we propose some techniques for processing multiple feature queries on separate index structures. Section 5 evaluates the results obtained by the merge techniques used, and the conclusions are summarized in Section 6.

2 Related work

2.1 Content-based Image Retrieval

Several systems have been proposed in recent years in the framework of content-based retrieval, both for still images and video sequences.

The QBIC system [8] allows queries to be performed on shape, texture, color, both by example and by sketch using as target media images and video shots. It appears to require a substantial level of human interaction during database population for features that require the interpretation of the image semantics.

The VIRAGE Engine [9] expresses visual features as image primitives. Primitives can be very general or quite domain specific. The basic philosophy underlying this architecture is a transformation from the data-rich representation of explicit image pixels to a compact, semantic-rich representation of visually salient characteristics.

In [16] segmenting techniques of video clips are based on content analysis for identifying the shots and the transitions between different scenes.

In [5] color, shape and texture are represented by trees in which each level is a different detail level of the image for that feature. Not relevant images can be quickly discarded by examining only the higher tree levels.

In [17] and [12] the authors work directly with pixel of images: in [17] patterns are identified and occurrences of each pattern are counted; in [12] the vector representing the pixel linearization is compressed with a statistic approach.

In [7] the authors propose to represent color information of image's subregions and to store only the features differences between a region and its subregion using a so called inter-hierarchical distance (*IHD*).

In [13] additional levels of abstracted histograms are added between a low dimension index and the original image histograms improving the performance.

2.2 Indexing techniques

Indexing on feature vectors is performed by hierarchical data structures that split the feature space in sub-spaces which guarantee the whole space covering through clusters of limited size containing neighbour elements.

Several structures and techniques have been proposed, which share some common aspects: first, all leaves are at the same tree level, and contain entries of the form (O_{id}, *point*), where O_{id} is a reference to the object in the database (the image), and *point* is the object descriptor (the feature vector). Second, an intermediate node contains entries of the form (*cp*, *area*), where *cp* is the address of a child node in the tree and *area* is the minimum area containing all the objects inside the child node; area representation may vary for different tree types. Third, every node and every leaf of the tree, except for the root, has a minimum and a maximum capacity.

The *R-tree* [10] is a dynamic organization in which the node entry's field *area* is represented by a (hyper-)rectangle with a minimum and a maximum bound for each dimension, and in which a volume minimization criteria is used for inserting new objects or splitting overfilled nodes. Decisions on which paths have to be traversed can be thus taken at higher tree levels.

The R^**-tree* [2] is a variant of *R-tree* which introduces also a region overlap minimization criteria during objects insertion. Furthermore during node splitting margin minimization is also applied, so that the rectangles will take a more square shape, thus improving store efficiency. These policies, together with a dynamic reorganization of the tree, lead to less split operations and a better spatial distribution of the tree entries. The QBIC system [8] uses an R^**-tree* indexing structure.

The *SS-tree* [19] is a dynamic organization in which the *area* field inside a node entry is represented with a *centroid* and a *radius*: this requires less memory space to store the area information, and increases node's capacity (therefore decreases tree's levels) speeding up the query processing. This structure provides, according to the authors, sensible improvements with respect to R^**-tree*. A new object is inserted by choosing the node with the closest centroid. Overfilled nodes are split by finding the dimension with the highest variance, and then by choosing the split position in order to minimize the sum of the variances on each side of the split. Like the R^**-tree*, this structure uses a forced reinsertion algorithm to dynamically reorganizing the tree.

[11] is an improvement based on a tighter bounding (hyper-)sphere than the one used in *SS-tree*, which also exploits the clustering property of the data by using a variant of the k-means clustering algorithm.

The *M-tree* [6] is a dynamic access method suitable for indexing generic "metric spaces", where the function to compute the distance between any two objects satisfies the positivity, symmetry and triangle inequality postulates. Similar to *SS-tree*, it defines a *routing object* instead of a centroid, and each entry stores the distance from its routing object. The pre-computed distances are used during query processing to reduce the number of nodes traversed, and the triangle inequality metric property accounts for less computation. Experimental results show that *M-tree* performs better than R^**-tree* on high dimensional vector spaces.

3 The VAMSplit R-tree indexing structure

The *VAMSplit R-tree* [18, 20] has a number of advantages over other index structures for static databases, e.g., image databases stored on CD-ROMs, whose content is fully known at indexing time. It performs better than *R^*-tree* and *SS-tree* in terms of query performance, time for creating the index, and space utilization, mostly because it minimizes or eliminates region overlap, and allows the tree nodes and leaves to be completely filled. The basic structure of *VAMSplit R-tree* is the same of *R-tree* and *R^*-tree*; it uses however a different algorithm, that recursively (1) determines the split dimension as the dimension of the dataset which has the maximum variance, (2) sorts the dataset with respect to that dimension, and (3) chooses a split point that is approximately the median point.

In [15] a *VAMSplit R-tree* implementation as a *C++* class library is described in detail. Several metrics are implemented, which can be selected by the user, and different searching techniques allow queries to be processed by considering both a whole global space for all the features, and separate indexing spaces, one for each feature, with query results merging. The following issues are faced:

- the tree is parametric on the feature space dimension;
- a disk block contains one node or one leaf (*bucket*), and disk block size resolves maximum node and leaf capacity automatically, according to parameters set by the user;
- the building process creates the tree in main memory and stores nodes and leaves on disk. The tree leaves are stored and deleted from main memory as they are created. The process generates two files, one for the true index, (i.e, the internal nodes), and one for the dataset (i.e., the leaves). Nodes are kept in main memory;
- retrieval is implemented by a *k-nearest neighbor strategy*; besides the target image, represented by its feature vectors, a query specifies the number k of images to retrieve, the metric to use for distance computation, and an approximation factor [18–20] which improves performance with a negligible effect on the retrieval accuracy.

Three metrics are implemented, L_1, L_2 and L_∞, between two points (when searching inside the tree leaves), and between a point and a hyper-rectangle (when searching inside the tree intermediate nodes). The distances D_1, D_2 and D_∞, between a point (i.e., a feature vector) $\mathbf{x}$ in the hyper-space and a hyper-rectangle with bounds $\{\mathbf{min},\mathbf{max}\}$, are defined as:

$$D_1(\mathbf{x},\{\mathbf{min},\mathbf{max}\}) = \sum_{k=1}^{n} d_k$$

$$D_2(\mathbf{x},\{\mathbf{min},\mathbf{max}\}) = \sqrt{\sum_{k=1}^{n} {d_k}^2}$$

$$D_\infty(\mathbf{x},\{\mathbf{min},\mathbf{max}\}) = max_{k=1\ldots n}\{d_k\}$$

where n is the feature space dimension, and d_k is defined as

$$d_k = \begin{cases} \mathbf{min}_k - \mathbf{x}_k & if\ \mathbf{x}_k < \mathbf{min}_k \\ 0 & if\ \mathbf{min}_k \le \mathbf{x}_k \le \mathbf{max}_k \\ \mathbf{x}_k - \mathbf{max}_k & if\ \mathbf{max}_k < \mathbf{x}_k \end{cases}$$

4 Multiple features image retrieval

While retrieving images by content similarity, the user evaluates the quality of a query result according to several features, e.g., color, shape, texture. The features usually bring different contributions to image similarity, and relevance feedback techniques [3, 4] can be used to tune the retrieval process by modifying the weight of different features in the global similarity evaluation.

We can devise two ways to cope with multiple features at index level:

1. *combined features*: one index tree for each combination of features is built, in which the vectors for each image feature are joined to make a single longer descriptor. This solution doesn't require any supplemental processing at query level, simply increasing the feature vector dimensions and the space required for storing the indexes.

2. *separate features*: a separate index is built and maintained for each feature. At query processing time the images returned by consulting each index separately must be merged in some way to obtain a single global rank. Several merge techniques could be used, as discussed later.

Combining the features together speeds-up the retrieval process but has several drawbacks, as we'll discuss in Section 5, the most serious coming from the different features value ranges that require an *a priori* normalization and do not allow the selection of different metrics or different weights.

By following a separate feature approach such drawbacks can be avoided, provided that the answers obtained by each index are merged in a correct way. Several merge algorithms can be, in principle, be used.

As a first solution, the sets of the images returned by considering each features separately could be simply intersected. The retrieval system would return to the user only the images that according to each features have the highest similarity score, i.e., the smallest distance. Beyond the difficulty of evaluating a threshold for the number of images to consider in each feature for getting a total of k-neighbor images, this solution can give good results only if the ranking in each feature are comparable.

A better solution is to build a global list containing all the images returned by single feature evaluation, ranked according some suitable function of the distances computed according to a single feature. The function should normalize the distance values of different features, and finally mediate the distances for the images returned in more than one feature evaluation, to obtain a single global distance for these one. The final rank in the list is assumed as the global image distance from the query, and the query result is the subset of the first k ranked images in the global list.

As for the distance normalization in each partial list, we propose two techniques:

norm1: for each element i in a ranked list of k elements having distance d_i from the query image, the normalized distance is $d_i^{'} = \frac{d_i}{d_k}$, where d_k is the lowest ranked element;

norm2: the normalized distance is $d_i^{'} = \frac{d_i - d_1}{d_k - d_1}$, where d_1 and d_k are respectively the highest and the lowest ranked element.

Norm1 is suitable when the returned image set is not satisfactory with respect to a single feature; it does not perform well if the retrieved images are similar to the query image, because the normalization increases the distance between the images with respect to a single feature, worsening the global result after the merge operation. **Norm2** solves this problem since it includes the images more similar to the query image with respect to the single feature result in the global result.

As for the combination of the partial distances into a global measure, we propose two mean distance functions:

mean1: if an image appears in n ranked lists, its global distance is

$$d = \frac{d_1 + d_2 + \ldots + d_n}{n}$$

where d_i are the distances of this image inside the different ranked lists in which appears;

mean2: the global distance is

$$d = \frac{d_1 + d_2 + \ldots + d_n}{n^2}$$

Mean1 gives good performance if an image is similar to the query image according to several features (hence it should appear in the global rank), but its global distance from the query image is greater than the distance of an image close to the query image according to only one feature. **Mean2** reduces the problem by decreasing the globally computed distance for an image when the number n of features according which it is relevant increases. Therefore, it shifts to higher (better) positions of the global rank the images contained in more feature ranked lists.

The normalization techniques and the mean distance functions are combined to provide four different merge techniques based on the same parametric algorithm shown in Figure 1. For retrieving the first k images most similar to a query image for n features, the algorithm's space complexity is $\Theta(n \cdot k)$ and the computational cost is $O(n^2 \cdot k^2)$.

5 Results and discussion

In this section we discuss the results obtained on some sample datasets. We have used a dataset of about 400 images picturing cars, aircrafts and ships, that was also used to evaluate feature extraction

```
for each entry e in ranked list rl[1]
  { compute the normalized distance e.d';
    add e to the global list gl
  }
for each j > 1
  for each entry e in ranked list rl[j]
    { compute the normalized distance e.d';
      used = false;
      for each entry g in the global list gl
        { if e.image == g.image
            { update g.d';
              sort gl;
              used = true;
              break
            }
        }
      if (not used) add e to the global list (sorted)
    }
return the first k entries gl
```

Fig. 1. Algorithm for the merge operation

and relevance feedback techniques [3, 4], using color, shape and texture features. The color feature is described by several vectors of 48 elements, representing the mean RGB, HSV, or HSB values in a 4 by 4 image segmentation. The shape is described by a vector of 200 elements representing the modified Fourier coefficients which describe the borders of the image objects. The texture is described by a vector of 8 elements representing the image's mean color value (1 value), the image's line directionality (3 values) and the image's granularity (4 values).

We have queried the dataset for combinations of color and shape, color and texture, and color, shape and texture together.

We have also used several datasets of randomly generated data simulating feature values of up to 1'000 and 10'000 elements, according to different clustered distributions, in order to test the time performance for building and querying the index. 10'000 is representative of the number of compressed images the can fill a CD-ROM, so it has been considered a plausible limit for a large class of image retrieval applications. Analytical results are described in [15].

A few tests performed by joining the features into a single global index have shown, as anticipated, that color similarity is overestimated. A suitable weighting of the index elements could solve or mitigate this problem. We proceeded instead in evaluating the use of separate indexes and the query result merging.

In general, the retrieval performance for separate features is good, because the different features can provide comparable contributions. The merge technique that gives the best results is the one based on **norm1** and **mean2** operations. This can be explained by observing that **norm1** normalization performs better than **norm2** when the images of a ranked list are different from the query but similar each other: in this case the final contribution after normalization is scarce, because the normalized distances approach the maximum value. **Mean2** computation provides better results than **mean1** because if an image appears in many more ranked lists, the computed mean distance is smaller, and the image rank will be higher in the global similarity list.

The retrieval efficiency is measured by comparing the construction time (Table 1) and the query statistics (Tables 2 and 3) of different parameter values: "Nds" and "Lvs" are respectively the average number of internal nodes and leaves accessed for a query; "t (ms)" is the average time in milliseconds for completing a query. The tests shown are referred to combined color and shape retrieval.

Table 1 shows that building a single tree for multiple features requires less time than building a tree for every feature. Considering the queries, combined features (Table 3) are faster than separate features (Table 2) because in the latter case it is necessary to search each index and then merge the results. We notice that for larger values of k, the performance of separate features vs. combined features indexing is worse, because the merge time grows polynomially in k. Moreover, the sum of leaves visited in different indexes for separate features is always equal or greater than the number of leaves visited for combined features.

Table 4 show the build time in milliseconds for random feature data on indexes of increasing size up to 10'000 elements.

color	*shape*	*texture*	*construction time*
●			196
	●		507
		●	115
●	●		627
●		●	210
●	●	●	656

Table 1. Build times of the trees measured in msec

		color			*shape*			*merge*
		L_1	L_2	L_∞	L_1	L_2	L_∞	
	Nds	3.9	3.5	2.8	8.5	5	4.3	
$k = 5$	Lvs	16.5	12.1	7	18.4	6.5	3.5	
	t (ms)	13.1	9.6	6.3	26.7	9.1	5.6	0.2
	Nds	4	3.9	3.3	10.1	6.2	5.1	
$k = 25$	Lvs	20.8	17.7	12.5	27.7	12.4	8.2	
	t (ms)	20.7	18.4	15.8	40.1	21.4	16.2	4.1
	Nds	4	4	3.9	11.1	7	5.8	
$k = 50$	Lvs	23.2	20.5	17.3	32.4	15.9	11.6	
	t (ms)	27.3	28.6	25.1	56.3	32.5	26.9	15.2

Table 2. Statistics for separate color and shape

metric	*k*	Nds	Lvs	t (ms)
L_1	5	16	29.2	41.3
L_∞	25	10.9	18.2	33.6
L_2	50	17.5	36.8	68.5

Table 3. Statistics for combined color and shape

n	100	500	1'000	5'000	10'000
build time	37	327	807	6372	15172

Table 4. Build time for large indexes measured in msec

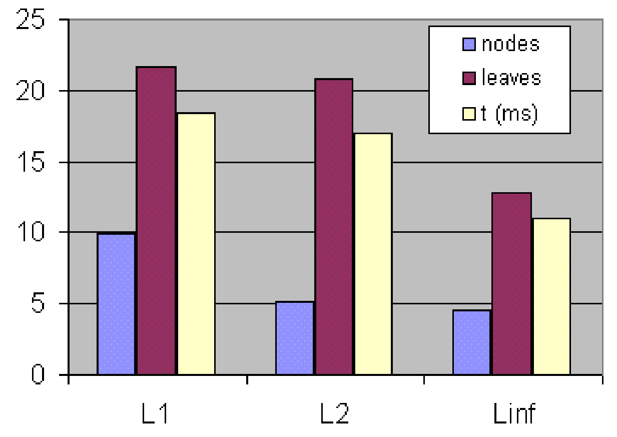

Fig. 2. Statistics for $n = 1'000$ depending on different metrics

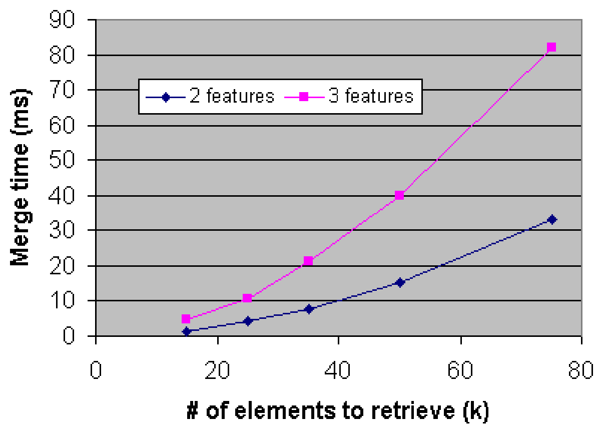

Fig. 3. Complexity of merge operation

Table 2 and Figure 2 show that both in the real dataset of 400 images and in randomly generated distributions the metric which provides better performance is L_∞, then L_2 and finally L_1; however, performance is inversely proportional to the retrieval accuracy, which is better with L_1, then with L_2 and finally L_∞ metric.

Figure 3 verifies, for the merge operation, the computational complexity estimated in Section 4, which is polynomial on the number n of features requested by the query and the number k of images to be retrieved.

In spite of the worse time performance of separate features with respect to combined features, keeping the features separate provides a number of important advantages:

– combining features requires a (new) tree to be built for each feature combination;
– they allow the user to specify a different metric for each index during the query. For some features different metric can provide sharper discrimination among images;
– in an iterative relevance feedback process unsatisfactory query results are easily to manage, because it is possible to change the metrics or the merge technique (e.g., by changing the normalization algorithm) in order to generate different results without the need of repeating the index search (as long as intermediate search results are stored in cache or temporary memory);
– since the ranked lists for each feature query are available, it is possible to inspect them separately, thus providing a good tuning tool during the development of specific image retrieval applications.

6 Conclusion

In this paper we have discussed the issue of building and querying high-dimension indexes representing image features for content based retrieval systems. We have discussed different approaches to the problem of querying multiple features. A good indexing structure for static databases (*VAMSplit R-tree*) and techniques for merging the query results have been applied to sample datasets, both real and simulated, in order to compare *combined features* versus *separate features* analysis. While combined features indexing provides more efficient results in term of computation time, separate features indexing shows acceptable time and space performance, and offers greater flexibility for iterative searches and relevance feedback analysis.

Promising directions for future work are (1) the extension of VAMSplit R-tree index structure for managing very large image databases, where the possibility of building the tree directly in secondary storage is important; (2) an implementation of a dynamic VAMSplit R-tree in which periodic reorganizations can be kept to a reasonable low level, and (3) the integration in the merge process of relevance feedback techniques.

References

1. M. Bartschi. An Overview of Information Retrieval Subjects. *IEEE Computer*, pages 67–84, May 1985.
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 *SIGMOD: International Conference on the Management of Data*, pages 322–331. ACM, May 1990.
3. A. Celentano and S. Chiereghin. Multiple Strategies for Relevance Feedback in Image Retrieval. Techn. Rep. CS-99-8, Department of Computer Science, Ca' Foscari University, Venice, 1999.
4. A. Celentano and E. Di Sciascio. Feature Integration and Relevance Feedback Analysis in Image Similarity Evaluation. *Journal of Electronic Images*, 7(2) 1998.
5. J. Y. Chen, C. A. Bouman, and J. P. Allebach. Multiscale Branch and Bound Image Database Search. In *SPIE: Storage and Retrieval for Image and Video Databases V*, pages 133–144, 1997.
6. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, pages 426–435, Athens, Greece, August 1997.
7. A. Dimai. Spatial Encoding using Differences of Global Features. In *SPIE: Storage and Retrieval for Image and Video Databases V*, pages 352–360, 1997.
8. M. Flickner et al. Query by Image and Video Content: The QBIC System. *IEEE Computer*, (28):23–31, September 1995.
9. A. Gupta and R. Jain. Visual Information Retrieval. *Communication of ACM*, (5):71–79, May 1997.
10. A. Guttman. R-trees: a Dynamic Index Structure for Spatial Searching. In *SIGMOD: International Conference on the Management of Data*, pages 47–57. ACM, 1984.
11. R. Kurniawati, J. S. Jin, and J. A. Shepherd. The SS^+-tree: An Improved Index Structure for Similarity Searches in a High-Dimensional Feature Space. In *SPIE: Storage and Retrieval for Image and Video Databases V*, pages 110–120, 1997.
12. R. Ng and A. Sedighian. Evaluating Multi-Dimensional Indexing Structures for Images Transformed by Principal Component Analysis. In *SPIE: Storage and Retrieval for Image and Video Databases IV*, pages 50–61, 1996.
13. R. Ng and D. Tam. An Analysis of Multi-level Color Histograms. In *SPIE: Storage and Retrieval for Image and Video Databases V*, pages 22–34.
14. J. Paredaens and B. Kuijpers. Data Models and Query Languages for Spatial Databases. *Data & Knowledge Engineering*, (25):29–52, 1997.
15. S. Sabbadin. Indexing Techniques for Image Retrieval Systems. Computer Science Degree Dissertation, Ca' Foscari University, Venice, Italy, 1999 (in Italian).
16. S. W. Smoliar and H. Zhang. Content-Based Video Indexing and Retrieval. *IEEE Multimedia*, pages 62–72, Summer 1994.
17. A. Soffer. Image Categorization Using $N \times M - grams$. In *SPIE: Storage and Retrieval for Image and Video Databases V*, pages 121–132, 1997.
18. D. A. White and R. Jain. Algorithms and Strategies for Similarity Retrieval. Available on the WWW at url *http://vision.ucsd.edu/papers/simret*.
19. D. A. White and R. Jain. Similarity Indexing with the SS-tree. Available on the WWW at url *http://vision.ucsd.edu/papers/sindex*.
20. D. A. White and R. Jain. Similarity Indexing: Algorithms and Performance. In *SPIE: Storage and Retrieval for Image and Video Databases IV*, pages 62–75, 1996. Also available on the WWW at url *http://vision.ucsd.edu/papers/sindexalg*.