Blobworld: A System for Region-Based Image Indexing and Retrieval (long version)*

Chad Carson, Megan Thomas, Serge Belongie,
Joseph M. Hellerstein, and Jitendra Malik

EECS Department, University of California, Berkeley, CA 94720, USA
{carson,mct,sjb,jmh,malik}@eecs.berkeley.edu

Abstract. Blobworld is a system for image retrieval based on finding coherent image regions which roughly correspond to objects. Each image is automatically segmented into regions ("blobs") with associated color and texture descriptors. Querying is based on the attributes of one or two regions of interest, rather than a description of the entire image. In order to make large-scale retrieval feasible, we index the blob descriptions using a tree. Because indexing in the high-dimensional feature space is computationally prohibitive, we use a lower-rank approximation to the high-dimensional distance. Experiments show good results for both querying and indexing.

1 Introduction

From a user's point of view, the performance of an information retrieval system can be measured by the quality and speed with which it answers the user's information need. Several factors contribute to overall performance:

- the time required to run each individual query,
- the quality (precision/recall) of each individual query's results, and
- the understandability of results and ease of refining the query.

All of these factors should be considered together when designing a system. In addition, image database users generally want to find images based on the *objects* they contain, not just low-level features such as color and texture [5, 7]; image retrieval systems should be evaluated based on their performance at this task.

Current image retrieval systems tend to perform queries quickly but do not succeed in the other two areas. A key reason for the poor quality of query results is that the systems do not look for meaningful image regions corresponding to objects. Additionally, the results are often difficult to understand because the system acts like a black box. Consequently, the process of refining the query may be frustrating. When individual query results are unpredictable, it is difficult to produce a stream of queries that satisfies the user's need.

* A shorter version of this paper will appear at the International Conference on Visual Information Systems '99 and is © Springer-Verlag.

In earlier work we described “Blobworld,” a new framework for image retrieval based on segmenting each image into regions (“blobs”) which generally correspond to objects or parts of objects [2, 3]. The segmentation algorithm is fully automatic, requiring no parameter tuning or hand pruning of regions. In this paper we present a complete online system for retrieval in a collection of 10,000 Corel images using this approach. The query system is available at `http://elib.cs.berkeley.edu/photos/blobworld`.

We present results indicating that querying for distinctive objects such as tigers, zebras, and cheetahs using Blobworld produces higher precision than does querying using color and texture histograms of the entire image. In addition, Blobworld false positives are easier to understand because the matching regions are highlighted. This presentation of results means that interpreting and refining the query is more productive with the Blobworld system than with systems that use low-level features from the entire image.

Because the speed of individual queries is also an important factor, we describe an approach to indexing Blobworld features in order to avoid linear scans of the entire image collection. We project each color feature vector down to a lower dimensional vector based on the Singular Value Decomposition [8] of the quadratic distance weight matrix and index the resulting vector. We find that queries that use the index to retrieve several hundred images and then rank those images using the full Blobworld algorithms achieve results whose quality closely matches the quality of queries that scan the entire database.

We begin this paper by briefly reviewing the current state of image retrieval. In Section 2 we outline the Blobworld segmentation algorithm, region descriptors, and querying system. In Section 3 we discuss indexing. In Section 4 we present experiments comparing the performance of Blobworld querying and indexing to the performance of a system that uses color and texture histograms of the entire image. We conclude with a brief discussion of our results.

1.1 Related Work

Many current image retrieval systems perform retrieval based primarily on low-level image features, including IBM's Query by Image Content (QBIC) [6], Photobook [19], Virage [9], VisualSEEk [23], Candid [15], and Chabot [18].

Lipson et al. [16] retrieve images based on spatial and photometric relationships within and across simple image regions. Little or no segmentation is done; the regions are derived from low-resolution images. Jacobs et al. [13] use multiresolution wavelet decompositions to perform queries based on iconic matching.

Ma and Manjunath [17] perform retrieval based on segmented image regions. Their segmentation is not fully automatic, as it requires some parameter tuning and hand pruning of regions.

Much research has gone into dimensionality reduction [10] and new index trees [22, 24] to cope with the high dimensionality of indices built over color histograms. Work to date has focused on indexing the entire image or user-defined sub-regions, not on indexing automatically created image regions. Our indexing methods are based on those used in QBIC [10].

2 Blobworld

The Blobworld representation is related to the notion of photographic or artistic scene composition. Blobworld is distinct from color-layout matching as in QBIC [6] in that it is designed to find objects or parts of objects; each image is treated as an ensemble of a few "blobs" representing image regions which are roughly homogeneous with respect to color and texture. A blob is described by its color distribution and mean texture descriptors. Figure 1 illustrates the stages in creating Blobworld. Details of the segmentation algorithm may be found in [2].

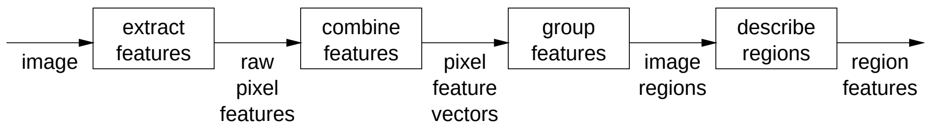


Fig. 1. The stages of Blobworld processing: From pixels to region descriptions.

2.1 Grouping pixels into regions

Each pixel is assigned a vector consisting of color, texture, and position features. The three color features are the coordinates in the L*a*b* color space [25]; we smooth these features to avoid oversegmentation arising from local color variations due to texture. The three texture features are contrast, anisotropy, and polarity, extracted at a scale which is selected automatically [2]. The position features are simply the (x, y) position of the pixel; including the position generally decreases oversegmentation and leads to smoother regions.

We model the distribution of pixels in this 8-D space using mixtures of two to five Gaussians. We use the Expectation-Maximization algorithm [4] to fit the mixture of Gaussians model to the data. To choose the number of Gaussians that best suits the natural number of groups present in the image, we apply the Minimum Description Length (MDL) principle [20, 21]. Once a model is selected, we perform spatial grouping of connected pixels belonging to the same color/texture cluster. Figure 2 shows segmentations of a few sample images.

2.2 Describing the regions

We store the color histogram over the pixels in each region. The histogram is based on bins with width 20 in each dimension of L*a*b* space. This spacing yields five bins in the L* dimension and ten bins in each of the a* and b* dimensions, for a total of 500 bins. However, not all of these bins are valid; only 218 bins fall in the gamut corresponding to $0 \leq \{R, G, B\} \leq 1$. (The other 282 bins are always empty.)

To match the color of two regions, we use the quadratic distance between their histograms $\mathbf{x}$ and $\mathbf{y}$ [10]:

$$d_{hist}^2(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^{\mathbf{T}}\mathbf{A}(\mathbf{x}-\mathbf{y})$$

where $\mathbf{A} = [a_{ij}]$ is a symmetric matrix of weights between 0 and 1 representing the similarity between bins i and j based on the distance between the bin centers; neighboring bins have a weight of 0.5. This distance measure allows us to give a high score to two regions with similar colors, even if the colors fall in different histogram bins.

For each blob we also store the mean texture contrast and anisotropy. The distance between two texture descriptors is defined as the Euclidean distance between their respective values of contrast and anisotropy$\times$contrast. (Anisotropy is modulated by contrast because it is meaningless in areas of low contrast.) We do not include polarity in the region description because it is generally large only along edges; it would not help us distinguish among different kinds of regions.

2.3 Querying in Blobworld

The user composes a query by submitting an image in order to see its Blobworld representation, selecting the relevant blobs to match, and specifying the relative importance of the blob features. We define an "atomic query" as one which specifies a particular blob to match (e.g., "like-blob-1"). A "compound query" is defined as either an atomic query or a conjunction of compound queries ("like-blob-1 and like-blob-2"). The score μ_i for each atomic query with feature vector $\mathbf{v}_i$ is calculated as follows:

1. For each blob b_j in the database image (with feature vector $\mathbf{v}_j$):
   (a) Find the distance between $\mathbf{v}_i$ and $\mathbf{v}_j$: $d_{ij} = (\mathbf{v}_i - \mathbf{v}_j)^{\mathbf{T}}\mathbf{\Sigma}(\mathbf{v}_i - \mathbf{v}_j)$.
   (b) Measure the similarity between b_i and b_j using $\mu_{ij} = e^{-\frac{d_{ij}}{2}}$. This score is 1 if the blobs are identical in all relevant features; it decreases as the match becomes less perfect.
2. Take $\mu_i = \max_j \mu_{ij}$.

The matrix $\mathbf{\Sigma}$ is block diagonal. The block corresponding to the texture features is an identity matrix, weighted by the texture weight set by the user. The block corresponding to the color features is the $\mathbf{A}$ matrix used in finding the quadratic distance, weighted by the color weight set by the user.

The compound query score for the database image is calculated using fuzzy-logic operations [14], and the user may specify a weight for each atomic query.

We then rank the images according to overall score and return the best matches, indicating for each image which set of blobs provided the highest score; this information helps the user refine the query. After reviewing the query results, the user may change the weighting of the blob features or may specify new blobs to match and then issue a new query. Figure 3 shows the results of a sample query.

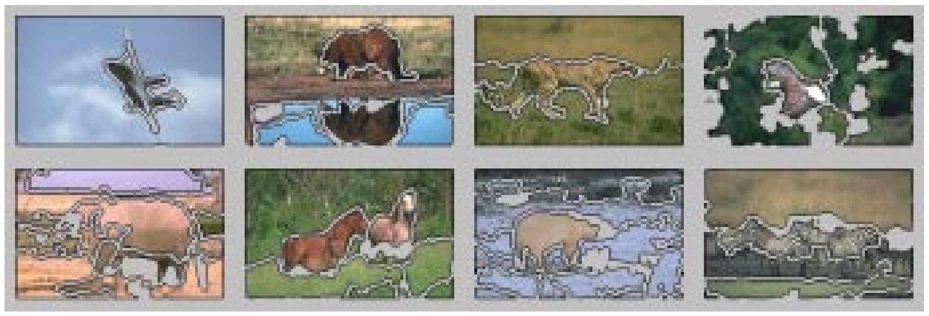
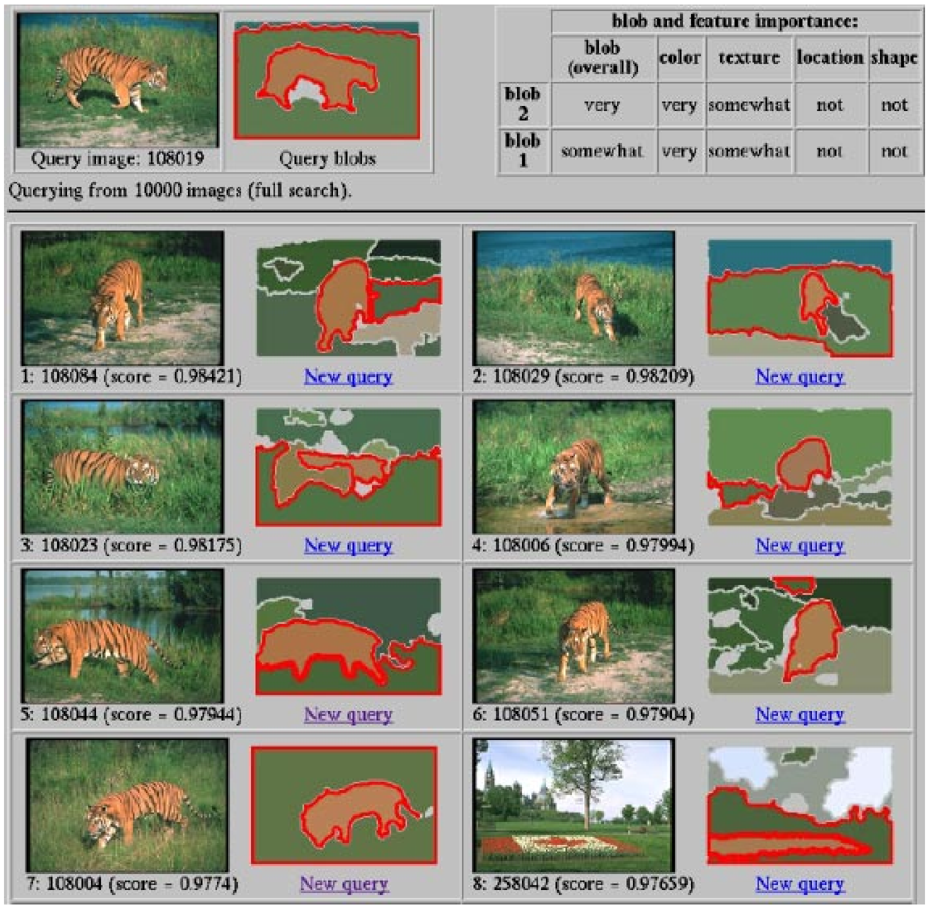


Fig. 2. Sample image segmentations.



	blob and feature importance:				
	blob (overall)	color	texture	location	shape
blob 2	very	very	somewhat	not	not
blob 1	somewhat	very	somewhat	not	not

Query image: 108019

Query blobs

Querying from 10000 images (full search).

1: 108084 (score = 0.98421) New query

2: 108029 (score = 0.98209) New query

3: 108023 (score = 0.98175) New query

4: 108006 (score = 0.97994) New query

5: 108044 (score = 0.97944) New query

6: 108051 (score = 0.97904) New query

7: 108004 (score = 0.9774) New query

8: 258042 (score = 0.97659) New query

Fig. 3. Results from a tiger query. The user selected the tiger blob and grass blob. Matching regions in each returned image are outlined in red.

3 Indexing

Indices allow the computer to find images relevant to a query without looking at every image in the database. We investigated indexing the color feature vectors to speed up atomic queries.

We used R*-trees [1], index structures for data representable as points in N-dimensional space. R*-trees are not the state of the art for nearest-neighbor search in multiple dimensions; using a newer tree [22, 24] would likely speed up our indexing results by a constant factor. However, our basic observations are independent of this tuning of the index: (i) indexing over blobs can decrease query time without significantly reducing quality, and (ii) indices over blobs can perform better than whole-image indices. As we tune and scale the system, we intend to examine new indexing schemes. We used the Generalized Search Tree framework [11] to experiment with the indices.

R*-trees break the multi-dimensional data space into successively smaller rectangles. Each node contains a list of the minimum bounding rectangles (MBRs) of its children and pointers to those children. The MBR of a node is a rectangle that minimally encloses all the children of that node. Leaf nodes contain all the data points enclosed within their MBRs.

Index speed degrades as the dimensionality of the data indexed increases. Because index trees are secondary storage structures, each node and leaf in an index tree must fit on a single disk page. Shorter paths from root to leaf in an index tree lead to fewer disk accesses to reach the leaves, and thus faster index retrievals. Node fanout, the number of data entries (MBR + pointer) that can fit in a node, dictates index tree height. Smaller data entries allow greater fanout and faster index retrievals. Higher dimensional data requires larger data entries and thus lower fanout. At sufficiently high dimensions fanout becomes so low that query speed using the index is worse than simply scanning the entire database. To avoid this, we need a low dimensional approximation to the full color feature vectors.

Computing the full distance $d(\mathbf{x}, \mathbf{y}) = \left[(\mathbf{x} - \mathbf{y})^{\mathbf{T}} \mathbf{A} (\mathbf{x} - \mathbf{y})\right]^{1/2}$ would require storing the entire 218-dimensional histogram and performing the full matrix-vector multiplication. To reduce the storage and computation in the index, we use Singular Value Decomposition to find $\mathbf{A}_k$, the best rank-k approximation to the weight matrix $\mathbf{A}$. We then project $\mathbf{x}$ and $\mathbf{y}$ into the subspace spanned by the rows of $\mathbf{A}_k$, yielding $\mathbf{x}_k$ and $\mathbf{y}_k$. The Euclidean distance $\left[(\mathbf{x}_k - \mathbf{y}_k)^{\mathbf{T}} (\mathbf{x}_k - \mathbf{y}_k)\right]^{1/2}$ is a lower bound on the full distance. Since the singular values $\sigma_{k+1}, \ldots, \sigma_{218}$ are small for our $\mathbf{A}$, this bound is tight. We can thus index the low-dimensional $\mathbf{x}_k$'s and use the Euclidean distance in the index without introducing too much error.

The index aims to match the quality of full queries without looking at all the blobs in the database. We would like the indices to retrieve exactly the images that the full query ranks as the best matches. Because the index does not use the full feature vector when selecting the images it returns, the index will not return images in exactly the same order as a full Blobworld query. There is a quality/time tradeoff: as the index returns more images, the final query results will get better, but the query will take longer.

4 Experiments

In order to understand the performance of the Blobworld system, we asked several questions:

- What is the precision of Blobworld queries compared to queries using global color and texture histograms? More specifically, for which queries does Blobworld do better, and for which do global histograms do better?
- How well do the indexing results approximate the results from a full scan of the collection? (According to this measure, false positives returned by the full scan should also be returned by the index.) Here we must consider the dimensionality of the indexed data as well as how indices over blobs compare to indices over whole image histograms.
- What do we lose by using an index instead of a full scan—how does the precision of the indexed query compare to the full-scan precision?

We explore each of these questions in turn in the next three sections.

4.1 Comparison of Blobworld and global histograms

We expected that Blobworld querying would perform well in cases where a distinctive object is central to the query. In order to test this hypothesis, we performed 50 queries using both Blobworld and global color and texture histograms.

We selected ten object categories: airplanes, black bears, brown bears, cheetahs, eagles, elephants, horses, polar bears, tigers, and zebras. There were 30 to 200 examples of each category among the 10,000 images.

We compared the Blobworld results to a ranking algorithm that used the global color and texture histograms of the same 10,000 images. The color histograms used the same 218 bins as Blobworld, along with the same quadratic distance. For texture histograms, we discretized the two texture features into 21 bins each. In this global image histogram case, we found that color carried most of the useful information; varying the texture weight made little difference to the query results.

For each category we constructed queries using one blob, two blobs, and global histograms. In each case we performed a few test queries to select the weights for color and texture and for each blob. We then queried using five new images. We used the same weights for each image in a category. We also were not allowed to choose whether to use one or two query blobs, which penalizes the Blobworld results; for example, in some bear images a background blob was salient to the category, while in others there was not a meaningful background blob. In Figure 4 we plot the average precision (fraction of retrieved images which are relevant) vs. recall (fraction of relevant images which are retrieved) for queries in the tiger, cheetah, zebra, and airplane categories; the differences between Blobworld and global histograms show up most clearly in these categories.

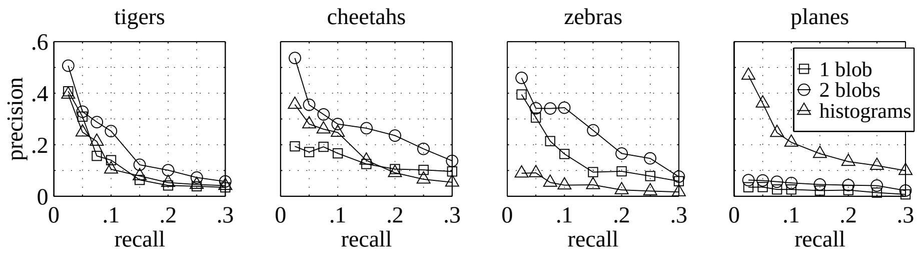


Fig. 4. Average precision (fraction of retrieved images which are relevant) vs. recall (fraction of relevant images which are retrieved) for Blobworld and global histogram queries. (Chance would yield precision ranging from 0.003 for zebras to 0.02 for airplanes.)

The results indicate that the categories fall into three groups:

distinctive objects: The color and texture of cheetahs, tigers, and zebras are quite distinctive, and the Blobworld result quality was clearly better than the global histogram result quality.

distinctive scenes: For most of the airplane images the entire scene is distinctive (a small gray object and large amounts of blue), but the airplane region itself has quite a common color and texture. Histograms did better than Blobworld in this category. (We have added an option to allow the user to use the entire background in place of a second blob. Using the background improves the Blobworld performance somewhat on these distinctive-scene queries, since it avoids matching, for example, the small regions of sky found in thousands of images in the database.)

other: The two methods perform comparably on the other six categories. Blobs with the same color and texture as these objects are common in the database, but the overall scene (a general outdoor scene) is also common, so neither Blobworld nor global histograms has an advantage, given that we used only color and texture. However, histograms can be taken no further, while Blobworld has much room left for improvement. For example, the shapes of animals and airplanes are quite distinctive. (Using the background also improves the Blobworld performance in some of these categories.)

These results support our hypothesis that Blobworld yields good results when querying for distinctive objects.

4.2 Comparison of Indexed to Blobworld Queries over Multiple Index Dimensionalities

Index speed improves as the number of dimensions in the index decreases; therefore, we want to find the minimum number of dimensions that the index may use. Query speed improves as the number of images retrieved by the index and then ranked by the full Blobworld algorithm decreases; therefore, we want to find the minimum number of images that the index must return. However, we must ensure that a query using the index produces nearly the same results as a full Blobworld query.

We are also interested in how well we can index blobs for Blobworld queries relative to how well we can index global feature vectors for global histogram queries.

For simplicity, we compare the indices based on color feature vectors alone.

We built indices over the color feature vectors of the blobs as well as over color feature vectors based on global histograms. We measured the recall of the indices using nearest-neighbor search [12] to retrieve and rank images against the top 40 images retrieved by a full Blobworld query or global histogram query over all the images.

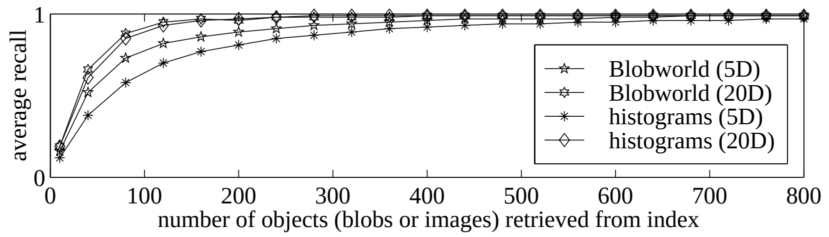


Fig. 5. Recall of (1) blob index compared to the top 40 images from the full Blobworld query and (2) global histogram index compared to the top 40 images from the full whole image query. The plots are the average of 200 queries over a database of 10,000 images, or about 61,000 blobs. Results using five dimensions and twenty dimensions are shown.

Figure 5 shows that in the low-dimensional case recall for the blob indices is higher than for the global histogram indices; blob indices approximate the results of full Blobworld queries better than global histogram indices approximate the results of global histogram queries. We believe this occurs because the blob color histograms, which are derived from relatively uniformly colored regions, cluster better than the global histograms.

Retrieving just a few hundred blobs from the five-dimensional blob index gives us most of the images the full Blobworld query ranked highest. Therefore, for the remaining experiments we use the five-dimensional indices.

4.3 Precision of indexed and full queries

The previous experiment showed that Blobworld indexing approximates the “true” Blobworld ranking more closely than global histogram indexing approximates the “true” histogram ranking. We also wanted to test the behavior of the indexing schemes in terms of precision measured against ground truth. In essence, we wanted to see how indexing affects the quality of Blobworld query results.

We performed the same queries as in Section 4.1, using the index to reduce the number of "true" comparisons required. We passed the query to an index using the five-dimensional projection and retrieved the nearest 400 database objects (400 blobs for Blobworld, 400 images for global histograms). When indexing two-blob queries, we retrieved the nearest 400 matches to each of the two blobs and returned the union of the two result sets. In all cases, we used the "true" matching algorithm to rank the retrieved images.

Figure 6 indicates that the precision of the indexed results closely mirrors the precision of the full query results. As previously stated, this is the quality goal for the index. Simple timing tests indicate that indexed Blobworld queries run in a third to half of the time of the full query. As the number of images in the collection increases, this speed advantage will become even greater.

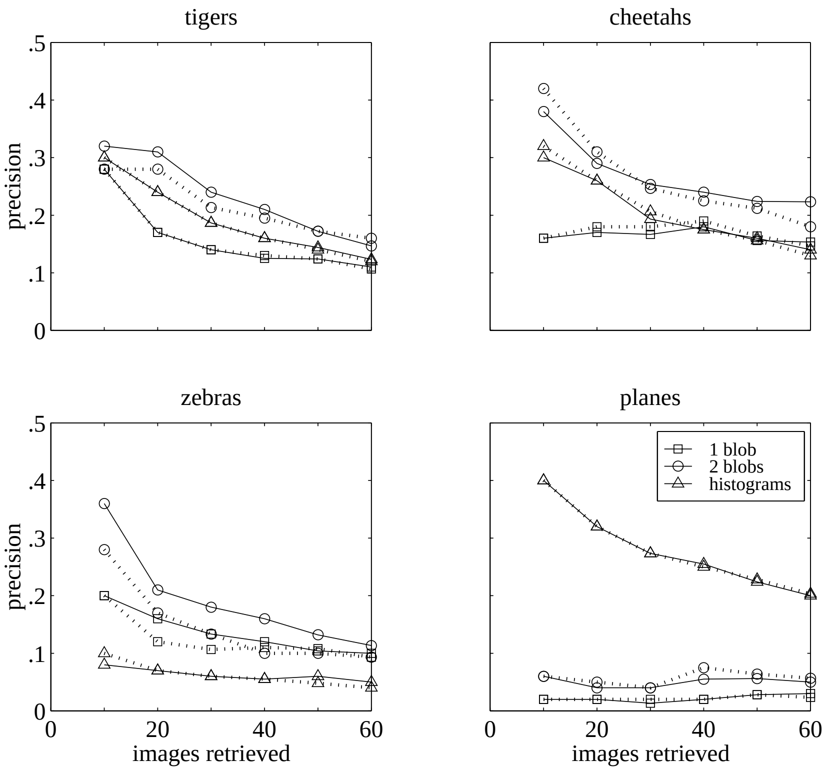


Fig. 6. Average precision (fraction of retrieved images which are correct) vs. number of images retrieved for several query types. Solid lines represent full queries; dashes represent indexed queries. The index tracks the corresponding full query quite closely, except for the zebra case, where the indexed Blobworld precision is lower than the full Blobworld precision because we only index color, not texture.

5 Conclusions

The effectiveness of an image retrieval system is determined by three factors: the time to perform an individual query, the quality of the query results, and the ease of understanding the query results and refining the query. We have shown that Blobworld queries for distinctive objects provide high precision and understandable results because Blobworld is based on finding coherent image regions. We have also shown that Blobworld queries can be indexed to provide fast retrieval while maintaining precision.

Acknowledgments

This work was supported by an NSF Digital Library Grant (IRI 94-11334), NSF graduate fellowships for Serge Belongie and Chad Carson, and fellowship stipend support for Megan Thomas from the National Physical Science Consortium and Lawrence Livermore National Laboratory.

References

1. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In Proc. ACM-SIGMOD Int'l Conf. on Management of Data (1990) 322–331
2. Belongie, S., Carson, C., Greenspan, H., Malik, J.: Color- and texture-based image segmentation using EM and its application to content-based image retrieval. In Proc. Int. Conf. Comp. Vis. (1998) http://www.cs.berkeley.edu/~carson/papers/ICCV98.html.
3. Carson, C., Belongie, S., Greenspan, H., Malik, J.: Region-based image querying. In IEEE Workshop on the Content-Based Access of Image and Video Libraries (1997)
4. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Statistical Soc., Ser. B, **39** (1977) 1–38
5. Enser, P.: Query analysis in a visual information retrieval context. J. Doc. and Text Management, **1** (1993) 25–52
6. Flickner, M., Sawhney, H., Niblack, W., Ashley, J., et al: Query by image and video content: The QBIC system. IEEE Computer, **28** (Sept. 1995) 23–32
7. Forsyth, D., Malik, J., Wilensky, R.: Searching for digital pictures. Scientific American, **276** (June 1997) 72–77
8. Golub, G., Loan, C. V.: Matrix Computations. Johns Hopkins University Press, 2nd edition (1989)
9. Gupta, A., Jain, R.: Visual information retrieval. Comm. Assoc. Comp. Mach., **40** (May 1997) 70–79
10. Hafner, J., Sawhney, H., Equitz, W., Flickner, M., Niblack, W.: Efficient color histogram indexing for quadratic form distance functions. IEEE Trans. Pattern Analysis and Machine Intelligence, **17** (July 1995) 729–736
11. Hellerstein, J. M., Naughton, J., Pfeffer, A.: Generalized search trees for database systems. In Proc. 21st Int. Conf. on Very Large Data Bases (1995) 562–573

12. Hjaltason, G., Samet, H.: Ranking in spatial databases. In Proc. 4th Int. Symposium on Large Spatial Databases (1995) 83–95
13. Jacobs, C., Finkelstein, A., Salesin, D.: Fast multiresolution image querying. In Proc. SIGGRAPH (1995)
14. Jang, J.-S., Sun, C.-T., Mizutani, E.: Neuro-Fuzzy and Soft Computing. Prentice Hall (1997)
15. Kelly, P., Cannon, M., Hush, D.: Query by image example: The CANDID approach. In SPIE Proc. Storage and Retrieval for Image and Video Databases (1995) 238–248
16. Lipson, P., Grimson, E., Sinha, P.: Configuration based scene classification and image indexing. In Proc. IEEE Comp. Soc. Conf. Comp. Vis. and Patt. Rec., (1997) 1007–1013
17. Ma, W., Manjunath, B.: NeTra: A toolbox for navigating large images databases. ACM Multimedia Systems Journal. (to appear)
18. Ogle, V., Stonebraker, M.: Chabot: Retrieval from a relational database of images. IEEE Computer, **28** (Sept. 1995) 40–48
19. Pentland, A., Picard, R., Sclaroff, S.: Photobook: Content-based manipulation of image databases. Int. J. Comp. Vis., **18** (1996) 233–254
20. Rissanen, J.: Modeling by shortest data description. Automatica, **14** (1978) 465–471
21. Rissanen, J.: Stochastic Complexity in Statistical Inquiry. World Scientific (1989)
22. Berchtold, D. K. S., Kriegel, H.: The x-tree: An index structure for high-dimensional data. In Proc. of the 22nd VLDB Conference (1996) 28–39
23. Smith, J. R., Chang, S.-F.: Single color extraction and image query. In Proc. IEEE Int. Conf. on Image Processing (1995) 528–531
24. White, D., Jain, R.: Similarity indexing with the ss-tree. In Proc. 12th IEEE Int'l Conf. on Data Engineering (1996) 516–523
25. Wyszecki, G., Stiles, W.: Science: Concepts and Methods, Quantitative Data and Formulae. Wiley, 2nd edition (1982)