Hierarchical Browsing and Search of Large Image Databases

Jau-Yuen Chen, Charles A. Bouman, *Senior Member, IEEE*, and John C. Dalton

***Abstract*—The advent of large image databases (>10 000) has created a need for tools which can search and organize images automatically by their content. This paper focuses on the use of hierarchical tree-structures to both speed-up search-by-query and organize databases for effective browsing.**

The first part of this paper develops a fast search algorithm based on best-first branch and bound search. This algorithm is designed so that speed and accuracy may be continuously traded-off through the selection of a parameter λ. We find that the algorithm is most effective when used to perform approximate search, where it can typically reduce computation by a factor of 20–40 for accuracies ranging from 80% to 90%.

We then present a method for designing a hierarchical browsing environment which we call a similarity pyramid. The similarity pyramid groups similar images together while allowing users to view the database at varying levels of resolution. We show that the similarity pyramid is best constructed using agglomerative (bottom-up) clustering methods, and present a fast-sparse clustering method which dramatically reduces both memory and computation over conventional methods.

***Index Terms*—Browse, image database, pyramids, search, trees.**

I. Introduction

In RECENT years, there has been a growing interest in developing effective methods for searching large image databases based on image content. The interest in image search algorithms has grown out of the necessity of managing large image databases that are now commonly available on removable storage media and wide area networks. The objective of this paper is to present hierarchical algorithms for efficiently organizing and searching these databases, particularly when they become large (>10 000).

Most approaches to image database management have focused on search-by-query [1]. These methods typically require that users provide an example image. The database is then searched for images which are most similar to the query. However, the effectiveness of search by query can be questionable [2]. First, it is often difficult to find or produce good query images, but perhaps more importantly, repetitive queries often tend to become trapped among a small group of undesirable images.

Browsing environments offer an alternative to conventional search-by-query, but have received much less attention. In general, a browsing environment seeks to logically and predictably organize the database so that users can find the images that they need.

Recently, several researches have applied multidimensional scaling (MDS) to database browsing by mapping images onto a two dimensional plane. MacCuish *et al.* [2] used MDS to organize images returned by queries while Rubner *et al.* [3] used MDS for direct organization of a database. However, existing MDS methods tend to be computationally expensive to implement and do not impose any hierarchical structure on the database.

Yeung *et al.* studied the application of clustering methods to the organization of video key frames [4], [5]. Their method is particularly interesting because it utilized complete-link agglomerative (bottom-up) clustering to organize the image key frames. The complete-link clustering was practical in this application since the number of key frames was relatively small. More recently, Milanes *et al.* have applied heirarchical clustering to organize an image database into visually similar groupings [6].

Zhang and Zhong [7] proposed a hierarchical self-organizing map (HSOM) which used the SOM algorithm to organize a complete database of images into a two-dimensional (2-D) grid. The resulting 2-D grid of clusters was then aggregated to form a hierarchy. An icon image was then assigned to each node to provide a useful browsing tool. However, a serious disadvantage of SOM is that it is generally too computationally expensive to apply to a large database of images.

In addition to being useful for search-by-query, fast search algorithms are often an essential component of image database organization and management. The typical search problem requires that one find the M best matches to a query image from a database of N images. Most search algorithms work by first extracting a K dimensional feature vector for each image that contains the salient characteristics to be matched. The problem of fast search then is equivalent to the minimization of a distance function $d(q, x_i)$ where q is the query image and x_i is the ith image in the database.

It has long been known that the computation of minimum distance search can be reduced when $d(\cdot,\cdot)$ has metric properties. For example, metric properties have been exploited to speed computation in vector quantization (VQ) [8], and key word search [9], [10]. More recently, such techniques have been applied to image database search [11], [12].

Manuscript received February 28, 1998; revised March 17, 1999. This work was supported by Apple Computer. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Tsuhan Chen.

J.-Y. Chen is with the Epson Palo Alto Laboratory, Palo Alto, CA 94306 USA (e–mail: jauyuen@erd.epson.com).

C. A. Bouman is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285 USA (e-mail: bouman@ecn.purdue.edu; http://www.ece.purdue.edu/~bouman/software/imgdatabase).

J. C. Dalton is with the Synthetik Software, San Francisco, CA 94103 USA.

Publisher Item Identifier S 1057-7149(00)01255-0.

Perhaps the most widely studied method for speeding minimum distance search is the k-d tree [13]–[15]. A k-d tree is constrained to have decision hyperplanes that are orthogonal to a coordinate axis. This orthogonal binary structure can severely limit the optimality of the tree particularly when $K \gg \log_2 N$, which is usually the case since, in practice, long feature vectors are important for achieving good matches.

One of the earliest treatments of hierarchical algorithms for fast search is by Fukunaga and Narendra [16]. This paper applies the triangle inequality to the problem of branch and bound search on tree structured clusterings formed using the K-means algorithm. Importantly, Fukunaga combines the branch and bound technique with depth-first search to locate the optimal solution. More recently, Roussopoulos *et al.* [17], White and Jain [18], and Kurniawati *et al.* [19] have considered similar approaches for image database search.

In this paper, we present the following tools for managing large image databases.

- A fast search algorithm which can perform exact search, or more importantly, can yield speed-ups of 20–40 for approximate search accuracies ranging from 80% to 90% [20] .
- A hierarchical browsing environment which we call a similarity pyramid that efficiently organizes databases so that similar images are located nearby each other [21].

The key to both these methods is the use of tree structured database organization. We discuss two distinct approaches to constructing these trees: top-down and bottom-up. While the top-down methods are well suited to the fast search problem, the bottom-up methods yield better results for the browsing application. Unfortunately, the conventional bottom-up methods require N^2 memory and computation. To address this problem we propose a fast-sparse clustering method which uses a sparse matrix of image distances together with the fast search algorithm to dramatically reduce both memory and computation requirements.

Our fast search algorithm is based on a best-first branch and bound search strategy.[1] The best-first search is in contrast to the depth-first branch and bound strategies of many previous studies [16]–[19]. The best-first approach is optimal in the sense that it searches the minimum number of nodes required to guarantee that the best match has been found. But perhaps more importantly, the best-first strategy results in excellent performance when an approximate bound is used to reduce search computation. In fact, we introduce a method to continuously trade-off search accuracy and speed through the choice of a parameter λ. For typical image search applications, this approximate search method yields a much better speed/accuaracy tradeoff than exact search methods.

Our browsing environment uses a similarity pyramid to represent the database at various levels of detail. Each level of the similarity pyramid is organized so that similar images are near by one another on a 2-D grid. This 2-D organization allows users to smoothly pan across the images in the database. In addition, different layers of the pyramid represent the database with varying levels of detail. At top levels of the pyramid, each image is a representative example of a very large group of roughly similar images. At lower levels of the pyramid, each image represents a small group of images that are very similar. By moving up or down the pyramid structure, the user can either zoom out, to see large variations in the database content, or zoom in, to investigate specific areas of interest. We propose a quality measure for the pyramid organization which we call *dispersion*, and use this measure to evaluate various approaches to pyramid design. Finally, we note that the image browsing environments described in this research can be adapted for specific search tasks through the use of relevance feedback [22], [23].

II. General Approach and Notation

Let the images in the database be indexed by $i \in S_0$ where S_0 is the complete set of N images. Each image will have an associated feature vector $x_i \in \mathbb{R}^D$ which contains the relevant information required for measuring the similarity between images. Furthermore, we will assume that the dissimilarity between two images i and j can be measured using a symmetric function $d(x_i, x_j)$. We will use a feature vector based on color, edge and texture image characteristics as described in the Appendix. More generally, the feature vectors could represent other data types, such as audio, but we will only consider images in this paper. Experimentally, we have found that an L_1 norm works well for many feature vectors, but we only assume that $d(x_i, x_j)$ is a metric, and therefore obeys the triangle inequality.

Tree structures will form the basis of both the search and browsing algorithms we will study. The tree structures will hierarchically organize images into similar groups, thereby allowing either a search algorithm or user to efficiently find images of interest. Let S denote the set of all tree nodes. Each node of the tree $s \in S$ is associated with a set of images $C_s \subset S$ and contains a feature vector z_s which represents the cluster of images. Generally, z_s will be computed as the centroid of the image features in the cluster. The number of elements in the cluster C_s will be denoted by $n_s = |C_s|$. The children of a node $s \in S$ will be denoted by $c(s) \subset S$. These children nodes will partition the images of the parent node so that

$$C_s = \bigcup_{r \in c(s)} C_r.$$

The leaf nodes of the tree correspond to the images in the database; so they are indexed by the set S_0. Each leaf node contains a single image, so for all $i \in S_0$, $z_i = x_i$ and $C_i = \{i\}$.

III. Top-Down and Bottom-up Clustering Algorithms

This section describes top-down and bottom-up methods for constructing trees. Each method will serve a useful role in database organization. While many well known methods exist for computing top-down trees, conventional bottom-up clustering methods require too much computation and memory storage to be useful for large databases of images.

In Section III-C we will introduce a fast-sparse clustering algorithm which implements the standard flexible agglomerative (bottom-up) clustering algorithm [24], but with dramati-

[1]The best-first branch and bound search is very similar to traditional $A*$ search, but uses a slightly different formulation of the cost functional.

cally reduced memory and computational requirements. We will then use the fast-sparse clustering algorithm to design the image browsing environments of Section V.

A. Top-Down Clustering

Top-down methods work by successively splitting nodes of the tree working from the root to the leaves of the tree [25]. We will use the K-means [26] or equivalently LBG [27] algorithms to partition each node of the tree into its K children, thereby forming a K^{ary} tree. The following describes the K-mean algorithm for partitioning the cluster $C \subset S$ into K sub-clusters $\{C_r\}_{r \in c(s)}$ using I iterations.

```
(1) Randomly select $K$ members $\{q_1, \cdots, q_K\}$ from $C$.
(2) For $I$ iterations {
    (a) For $r = 1$ to $K$,
        $C_r \leftarrow \{x \in C: d(x, q_r) < d(x, q_j) \text{ for } j \neq r\}$.
    (b) For $r = 1$ to $K$,
        $q_r \leftarrow \text{centroid } (C_r)$.
    }.
```

Here, we have allowed for a slight generalization of K-means in which the centroid can be computed in one of three possible ways: mean, median, and minimax.

$$\text{Mean:} \quad q_r = \frac{1}{|C_r|} \sum_{s \in C_r} x_s \tag{1}$$

$$\text{Median:} \quad [q_r]_i = \text{median}_{s \in C_r}([x_s]_i) \tag{2}$$

$$\text{Minimax:} \quad q_r = \arg\min_q \left\{ \max_{s \in C_r} d(q, x_s) \right\} \tag{3}$$

Here, $[q_r]_i$ refers to the ith component of the vector q_r. Top-down tree growing using the K-means algorithm is quite efficient since each cluster is split independently of the others. However, the tree is usually too deep to be useful for browsing. To enforce a balanced tree, we add a criterion, $|C_r| \leq \lceil (N/K) \rceil$, to step 2(a). For a balanced tree, the computation is of order NKI where N is the number of pixels, K is the number of splits per node, and I is the number of iterations. Notice that this is linear in N.

B. Bottom-Up Clustering

While top-down clustering is fast, it tends to produce poor clusterings at lower levels of the tree. This is not surprising since once an image which is placed in an undesirable cluster it is constrained to remain in that branch of the tree. For this reason, bottom-up clustering seems to offers superior performance for browsing applications [28].

Conventional bottom-up (agglomerative) clustering algorithms work by first forming a complete matrix of distances between the images and then using this matrix to sequentially group together elements [25], [29]. Let $C_i = \{i\}$, be the disjointed clusters each of size n_i. The proximity matrix $[d_{ij}]$ defines the pairwise distances between clusters i and j. Initially, the proximity matrix is set equal to the distances between the images x_i and x_j. Each iteration of agglomerative clustering combines the two clusters, i and j, with the minimum distance. The new cluster formed by joining i and j is denoted by k, and the distance from k to each of the remaining clusters is updated.

Since the distance matrix is assumed symmetric, $d_{ij} = d_{ji}$ is a symmetric matrix, and only its upper triangular component need be stored. In order to simplify notation, we will assume that the notation d_{ij} refers to the unique entry given by $d_{\min(i,j),\max(i,j)}$. Using these conventions, the general algorithm for agglomerative clustering has the following form.

```
1) $S \leftarrow \{0, 1, \cdots, N-1\}$;
2) For each $(i,j) \in S^2$ compute $d_{ij} \leftarrow d(x_i, x_j)$;
3) For $k = N$ to $2N-2$ {
    (a) $(i^*, j^*) = \arg\min_{(i,j) \in S^2} d_{ij}$
    (b) Set $C_k \leftarrow C_{i^*} \cup C_{j^*}$ and $n_k \leftarrow n_{i^*} + n_{j^*}$
    (c) $S \leftarrow \{S - \{i^*\} - \{j^*\}\} \cup \{k\}$
    (d) For each $h \in S - \{k\}$,
        compute $d_{hk} \leftarrow f(d_{hi^*}, d_{hj^*}, d_{i^*j^*}, n_h, n_i^*, n_j^*)$
   }.
```

The specific type of agglomerative clustering is defined by the choice of the function $f(\cdot)$ in step 3(d) of the algorithm. Lance and Williams proposed the following general functional form for $f(\cdot)$ because it includes many of the most popular clustering methods [24]

$$d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} - \gamma |d_{hi} - d_{hj}|. \tag{4}$$

Here, α_i, α_j, β, and γ are coefficients which depend on some property of the clusters. Table I lists the particular choices of these coefficients for a number of standard clustering algorithms [25], [29].

Some clustering methods are said to be dilating if individual elements not yet in groups are more likely to form nuclei of new groups. Although this tends to produce "nonconformist" groups of peripheral elements, dilating methods have the advantage that they produce more balanced trees. So for example, complete link clustering is dilating, and therefore tends to create balanced trees; while single link clustering is known to create very deep, unbalanced trees.

We will focus our attention on the flexible clustering algorithm of Lance and Williams [24] which uses the update rule

$$d_{hk} = \frac{1-\beta}{2} d_{hi} + \frac{1-\beta}{2} d_{hj} + \beta d_{ij} \tag{5}$$

where β is a parameter taking on values in the set $-1 \leq \beta \leq 1$. We are particularly interested in the case $\beta = -1$ since this is the maximally dilating case which generates the most balanced trees. For this case

$$d_{hk} = d_{hi} + d_{hj} - d_{ij}. \tag{6}$$

C. Fast-Sparse Clustering

For a database with N images, standard agglomerative clustering requires the computation and storage of an $N \times N$ proximity matrix, $[d_{ij}]$. For most image database applications, this is unacceptable. In order to reduce memory and computation requirements, we propose a fast-sparse clustering algorithm based on the flexible clustering of the previous section. This method uses a sparse proximity matrix containing the distances between each image, i, and its M closest matches. The M closest matches will then be computed using the fast search algorithm of Section IV.

TABLE I
TABLE OF RECURSION COEFFICIENTS FOR STANDARD PAIRWISE AGGLOMERATIVE CLUSTERING ALGORITHMS

Clustering method	α_i	α_j	β	γ	Effect on space
Single Link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	Contracting
Complete Link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	Dilating
UPGMA	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0	Conserving
WPGMA	$\frac{1}{2}$	$\frac{1}{2}$	0	0	Conserving
UPGMC	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i n_j}{(n_i+n_j)^2}$	0	Conserving
WPGMC	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0	Conserving
Ward's	$\frac{n_i+n_h}{n_i+n_j+n_h}$	$\frac{n_j+n_h}{n_i+n_j+n_h}$	$\frac{-n_h}{(n_i+n_j+n_h)}$	0	Dilating
Flexible	$\frac{1-\beta}{2}$	$\frac{1-\beta}{2}$	β	0	Contracting if $\beta \geq 0$, Dilating if $\beta < 0$

For each image i, we will only store the entries d_{ij} where j is one of the M best matches to image i. This will form a sparse matrix with NM entries. The disadvantage of this sparse matrix is that the conventional update equation of (6) can no longer be applied because of the missing entries. For example, when combining clusters i and j to form the new cluster k, we must recompute d_{hk} the distance between k and every other cluster h. Unfortunately, the update equation of (6) can not always be computed because the terms d_{hi} and d_{hj} may be missing from the sparse matrix. Note that the term d_{ij} must be available since it is chosen as the minimum distance term in the matrix. In order to address this problem, we will replace the missing terms with the estimates $\hat{d}_{hi}$ and $\hat{d}_{hj}$ when needed. The new update rule then becomes

$$d_{hk} = \begin{cases} d_{hi} + d_{hj} - d_{ij}, & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are available} \\ \hat{d}_{hi} + d_{hj} - d_{ij}, & \text{if } d_{hi} \text{ is missing} \\ d_{hi} + \hat{d}_{hj} - d_{ij}, & \text{if } d_{hj} \text{ is missing} \\ \text{Do not compute}, & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are missing.} \end{cases} \tag{7}$$

Notice that if both entries are missing, then the updated distance is not entered into the sparse matrix.

The question remains of how to compute an estimated entry $\hat{d}_{hi}$. Consider a modified clustering algorithm which uses the recursion

$$\tilde{d}_{hk} = \tilde{d}_{hi} + \tilde{d}_{hj} \tag{8}$$

in place of the recursion of (6). For the same clusters h and k, this modified recursion would over estimate the true distance, i.e., $d_{hk} < \tilde{d}_{hk}$. However, this recursion has the advantage that the solution may be expressed in closed form

$$\tilde{d}_{hk} = \sum_{m \in C_h} \sum_{n \in C_k} d(x_m, x_n). \tag{9}$$

We may use (9) to approximate the missing term $\hat{d}_{hi}$ of (7). We do this in terms of the quantity r_i, the distance between image i and its Mth closest match. More specifically, let $\pi(i,j)$ be the index of the jth closest match to image i. Then the distance to the Mth closest match is

$$r_i = d(x_i, x_{\pi(i,M)}).$$

The required lower bound is then given by

$$\begin{aligned} \tilde{d}_{hi} &= \sum_{m \in C_h} \sum_{n \in C_i} d_{mn} \\ &\geq \sum_{m \in C_h} \sum_{n \in C_i} \max(r_m, r_n) \\ &\geq \max\left(\sum_{m \in C_h} \sum_{n \in C_i} r_m, \sum_{m \in C_h} \sum_{n \in C_i} r_n \right) \\ &= \max\left(n_i \sum_{m \in C_h} r_m, n_h \sum_{n \in C_i} r_n \right) \end{aligned}$$

where the second inequality results from Jensen's equality and the convexity of the $\max(\cdot,\cdot)$ function, and n_i is the number of elements in cluster i. This yields the final approximation

$$\hat{d}_{hi} \triangleq \max\left(n_i \sum_{m \in C_h} r_m, n_h \sum_{n \in C_i} r_n \right) \tag{10}$$

where the summation terms may be recursively computed as part of the clustering algorithm. Fig. 1 shows the complete sparse clustering algorithm where S_i is the set of sparse entries of each image i, and the set $S \times S_i = \{(i,j): i \in S \text{ and } j \in S_i\}$. Section IV will define a fast algorithm for computing the M closest matches to an image. This algorithm can be used to efficiently compute the entries of the sparse matrix. We note that the fast search algorithm of Section IV requires the construction of a top-down tree. However, since the top-down tree design can be done very efficiently, this does not present an excessive computational overhead. The general approach to fast-sparse clustering is then

1) construct a tree using the top-down method of Section III-A;
2) construct the sparse distance matrix using the fast search algorithm of Section IV together with the top-down tree;
3) apply the sparse clustering algorithm of Fig. 1.

```
1. $S \leftarrow \{0, 1, \cdots, N-1\}$
2. For each $i \in S$ {
   (a) $r_i \leftarrow d(x_i, x_{\pi(i,M)})$
   (b) $S_i \leftarrow \{\pi(i,1), \cdots, \pi(i,M)\}$
   (c) For $j \in S_i$,
           compute $d_{ij} \leftarrow d(x_i, x_j)$
   }
3. For $k = N$ to $2N-2$ {
   (a) $(i^*, j^*) = \arg \min_{(i,j) \in S \times S_i} d_{ij}$
   (b) Set $C_k \leftarrow C_{i^*} \cup C_{j^*}$; $n_k \leftarrow n_{i^*} + n_{j^*}$; $r_k \leftarrow r_{i^*} + r_{j^*}$
   (c) $S \leftarrow \{S - \{i^*\} - \{j^*\}\} \cup \{k\}$
   (d) $S_k \leftarrow S_{i^*} \cup S_{j^*}$
   (e) For each $h \in S_k$,
           apply equation (7) with $\hat{d}_{hi} = \max(n_h r_h \, , \, n_i r_i)$
   }
```

Fig. 1. Algorithm used for flexible clustering with sparse distance matrix.

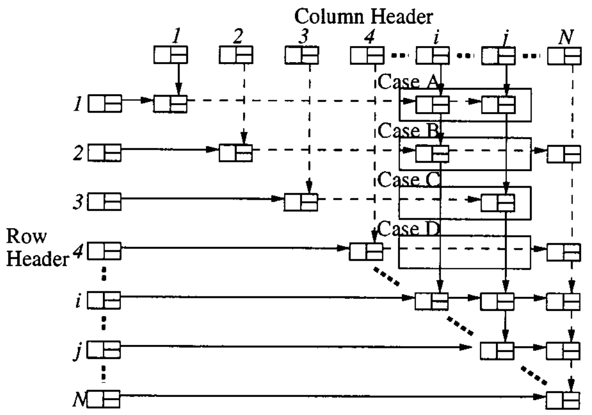


Fig. 2. Data structure used for storing sparse matrix for clustering. Both rows and columns have the structure of linked lists. Dark lines are used for links along columns, while doted lines link rows. Each case represents one of the four update possibilities for (7).

Fig. 2 illustrates the sparse matrix data structure that we use for the clustering. The sparse matrix is upper triangular and has a linked list structure along both its rows and columns. Both insertions and deletions of elements can be made as with a conventional linked list, but with link pointers along both rows and columns. For any specific image i, the stored values of d_{ij} can be found by scanning down the ith column and then across the ith row. We will refer to this list of entries as the ith row-column. In practice when clusters i and j are merged (assume $i < j$), the new cluster k is stored along the row-column previously containing cluster i, and the row-column corresponding to j is removed. The figure illustrates the four possible cases required for the update of (7). For case A, both elements are present. The result of (7) is stored in d_{2i}, and the entry d_{1j} is deleted. For case B, the element d_{1j} is missing; so it must be computed using $\hat{d}_{2j}$. The result of (7) is then stored in d_{2i}. For case C, a new entry must be inserted into location d_{3i}, and the entry at d_{3j} must be removed. For case D, nothing is done.

D. Binary to Quadtree Transformation

One limitation of pairwise clustering algorithms is that they can only generate binary tree structures. In principle it is possible to generate K^{ary} trees, but the algorithms to do this would be of order N^K which is not acceptable for our application. Therefore, our approach is to map the binary trees to K^{ary} trees in a manner that minimizes a cost criteria. We are particularly interested in the quadtree case ($K = 4$) since this will be important for the browsing application of Section V.

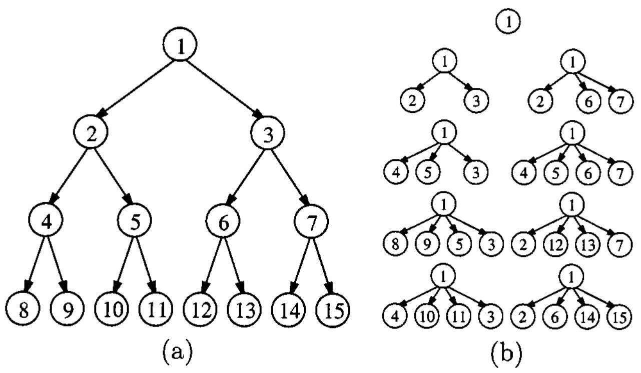


Fig. 3. All four partitions of a binary tree node $s = 1$. (a) Original binary tree with root node $s = 1$. (b) The nine unique partitions of the node $s = 1$. Each partition contains four or less nodes that partition the original node.

Let $P \subset S$ be a set of nodes in the binary tree. We say that P is a K partition of C_s if P contains K nodes, and

$$C_s = \bigcup_{r \in P} C_r$$
$$\emptyset = C_r \cap C_l \quad \text{for all } r, l \in P \text{ with } r \neq l.$$

Let $\mathbf{P}(s, K)$ denote the set of all possible partitions with K **or less** nodes

$$\mathbf{P}(s, K) = \{P\text{: } P \text{ is a partition of } s \text{ with } K \text{ or less nodes}\}.$$

Fig. 3 shows all four partitions of a binary tree node labeled as $s = 1$. Each of the nine possible partitionings contains four or less nodes which contain all of the elements in the root node.

The binary tree may then be transformed to a K^{ary} tree by working from the tree's root to its leaves. Each node s is directly connected to the optimal set of partition nodes P^* which minimize a cost function subject to the constraint that $P \in \mathbf{P}(s, K)$. In order to minimize tree depth, we use the maximum number of elements in the children nodes as the cost function. That is

$$P^* = \min_{P \in \mathbf{P}(s,K)} \left\{ \max_{j \in P} n_j \right\} \tag{11}$$

where $n_j = |C_j|$.

IV. Fast Search

Fast search is a basic need in applications that manage large image databases. The general problem is to find the M best matches to a query image (or feature vector) provided by a user. The objective of these algorithms is to find these M best images in less time than is required to do a full linear search of the database. Such query activities are likely to be an essential task for managing large image databases, but in addition we showed in Section III-C that the fast-sparse clustering algorithm requires fast search to efficiently construct the sparse matrices.

In the following section, we present a method for either approximate or exact search which exploits a hierarchical tree

```
1. $s^* = \text{root}$
2. $\Omega = \{s^*\}$
3. NodesSearched $\leftarrow 0$
4. While $s^*$ is not a leaf node {
  (a) $\Omega \leftarrow (\Omega - \{s^*\}) \bigcup c(s^*)$
  (b) NodesSearched $\leftarrow$ NodesSearched $+ |c(s*)|$
  (c) $s^* \leftarrow \arg\min_{s \in \Omega} \underline{d}_s$
  }
```

Fig. 4. Algorithm used for both best-first search, and branch and bound search.

structure. Our method couples a best-first implementation of branch and bound search with the structure imposed by a top-down tree of Section III-A to substantially reduce computation.

A. Best First Search

Standard search methods for tree structures transverse directly from the root to a leaf of the tree choosing the branch which minimizes distance between the query, q, and the cluster centroid, z_s, at each point. However, this search strategy is not optimum since it does not allow for back tracking.

Fig. 4 shows the algorithm used for best first search of trees [30]. Best first search works by keeping track of all nodes which have been searched, and always selecting the node with minimum cost to search further. While best-first search is greedy, it does allow for backtracking. For the moment, we do not specify how the cost, $\underline{d}_s$, is computed for internal nodes of the tree. However, for leaf nodes it will always be computed as $\underline{d}_i = d(q, x_i)$. The minimization within the while loop of Fig. 4 may be efficiently implemented using a data structure known as a heap [31] to keep track of the smallest current cost node. Insertions and deletions from a heap can be done in $n \log n$ time where n is the number of entries. However, heaps can be very efficiently implemented, so for practical cases the computational cost is dominated by the evaluation of the cost functions $\underline{d}_s$. Therefore, the total computation is assumed proportional to the variable NodesSearched.

The extension for K-nearest neighbor search is quite straightforward. Since the best first search allows backtracking, we can simply change the exit condition of while loop from the first leaf node to the kth leaf node.

B. Branch and Bound Search

Under certain conditions best-first search is guaranteed to find the minimum cost solution. More specifically, if $\underline{d}_s$ is a lower bound on the cost of all leaf nodes that are its descendants, then this is branch and bound search [30] and the solution is guaranteed to be the global minimum. Then $\underline{d}_s$ must have the property that

$$\underline{d}_s \le \min_{i \in C_s} d(q, x_i).$$

A loose lower bound will result in a full search of the tree, but a tight lower bound will result in an efficient search which follows a direct path to the optimum node with little backtracking.

Fukunaga and Narendra suggested using the triangle inequality to the distance metric $d(q, z_s)$ to provide the required lower bound [16]. Define the radius of node s as

$$R_s = \max_{i \in C_s} d(x_i, z_s)$$

where z_s is the code word associated with node s and x_i is the feature vector of an image contained in leaf node i. Then for all $i \in C_s$

$$\begin{aligned} d(q, x_i) &\ge d(q, z_s) - d(x_i, z_s) \\ &\ge d(q, z_s) - R_s. \end{aligned}$$

Using this inequality, we propose a general form for the cost function $\underline{d}_s$

$$\underline{d}_s = d(q, z_s) - \lambda R_s$$

where $0 \le \lambda \le 1$ is a constant.

When $\lambda = 1$ the search is guaranteed to yield the optimum solution. However, this bound is often too conservative. A smaller value of λ is often desirable since it can dramatically reduce the number of nodes to be searched while retaining good accuracy. We will see that that this form of approximate search is very useful and yields in a much better accuracy/speed tradeoff then existing approximate search methods such as epsilon search [32].

When $\lambda < 1$ the accuracy can be improved by searching for more images than are required and then selecting the best from those that are returned. For example, if the user requests the ten most similar images, these ten images can be selected from the 20 best images found using the approximate branch and bound search. This strategy can slightly improve the accuracy/computation tradeoff.

V. Database Browsing

In this section, we propose a structure which we call a similarity pyramid that allows users to move through the database in a natural manner that is analogous the the evolution of computer search algorithms such as branch and bound. The similarity pyramid is created by mapping each level of a quadtree onto an 2-D grid to form a level of the pyramid. The pyramid differs from the quadtree in two important respects. First, each node of the pyramid represents a specific location on a 2-D grid. Therefore, the data can be presented to a user as a flat 2-D array of objects, allowing the user to pan across objects at a fixed level of the pyramid. Second, the position of elements at different levels of the pyramid have a well defined spatial registration. This allows users to move up and down through levels of the pyramid while retaining relative orientation in the database.

Fig. 5 illustrates a simple example of a similarity pyramid and its associated quadtree. Fig. 5(a) shows three levels of a quadtree corresponding to $l = 0$, 1 and 2. Each node represents a cluster of images in the database with the leaf nodes representing individual images. Fig. 5(b) shows level $l = 1$ of the similarity pyramid while Fig. 5(c) shows level $l = 2$. At level $l = 1$, each cluster is represented by a single icon image chosen from the cluster. For node s, we constrain the icon image to be one of the four corresponding icon images at $l = 2$. This is useful because

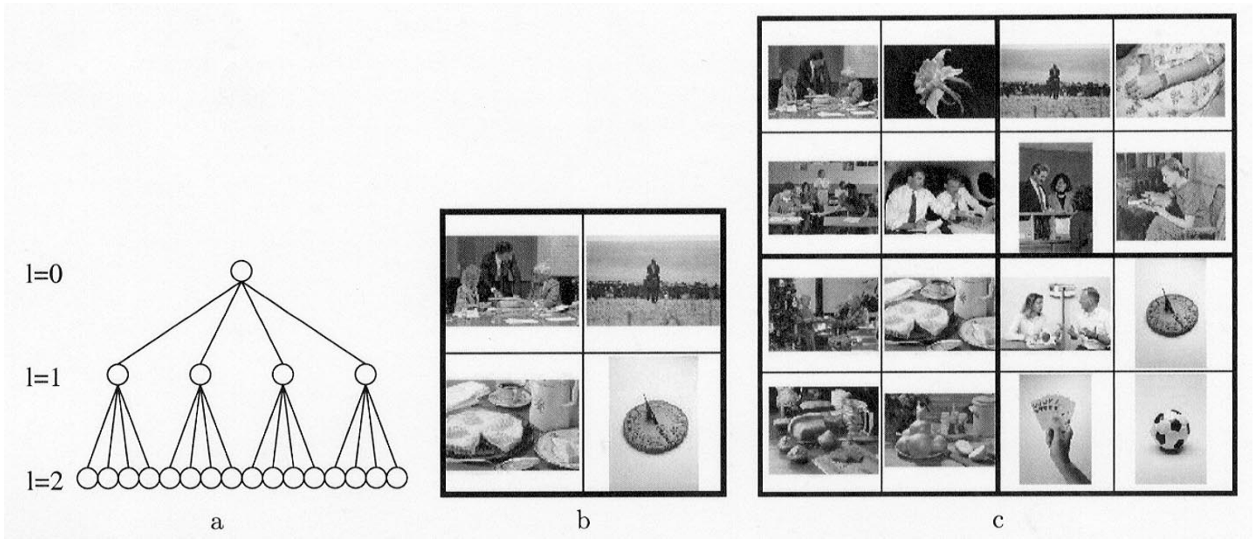


Fig. 5. This figure illustrates how nodes are mapped from the quadtree to the similarity pyramid. There are 24 possible mappings of four children nodes to four pyramid nodes. The mapping is chosen to maximize spatial smoothness in image characteristics.

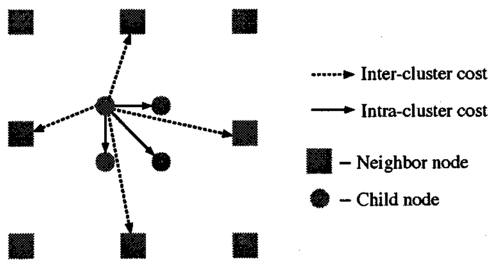


Fig. 6. This figure illustrates the inter-cluster and intra-cluster cost terms used to organize images in the pyramid. The neighbor nodes are at level l of the pyramid, while the children nodes are at level $l-1$.

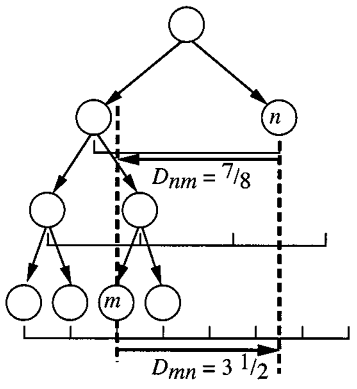


Fig. 7. This figure illustrates the distance measured between two images in the similarity pyramid. The distance is an asymmetric function because it is measured relative to the first image.

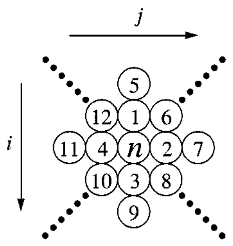


Fig. 8. An example of a dense packing of the M closest matches to image n. The M images are arranged in a diamond shape such that the physical distance $D_{n\pi(n,m)}$ is nondecreasing with respect to m.

it allows a user to keep relative orientation when moving between levels of the pyramid. The specific icon image is chosen to minimize the distance to the corresponding cluster centroid, z_s where the cluster centroid is computed using the mean computation of (1).

Notice that the mapping of the quadtree to the pyramid is not unique since each group of four child nodes can be oriented in $24 = 4!$ distinct ways.

Fig. 13 shows an example application for browsing through the similarity pyramid. For a large database, even upper levels of the pyramid will be too large to display on a single screen. Therefore, the user can move along the x or y directions in a panning motion to search for image clusters of interest. If a specific cluster appears to be of interest, then the user can choose to move down to the next level by "double clicking" on a cluster. The next level of the pyramid is then presented with the corresponding group of four children clusters centered in the view. Alternatively, the user may desire to backtrack to a higher level of the pyramid. In this case, the previous level of the pyramid is presented with proper centering.

We will primarily use the fast-sparse clustering algorithm of Section III-C to build the similarity pyramid. The fast-sparse clustering algorithm requires computation comparable to top-down clustering algorithms, but we have found that it produces better results. This is because top-down pyramids tend to place a substantial number of images into inappropriate clusters midway down the quadtree. These misplaced images are effectively lost in the same manner that a miss-filed book may be lost in a large library. Since bottom-up pyramids have better clustering at lower levels, this miss-filing effect is substantially reduced.

A. Quadtree to Pyramid Mapping

In this section, we describe our method for mapping nodes of the quadtree to nodes of the similarity pyramid. This mapping

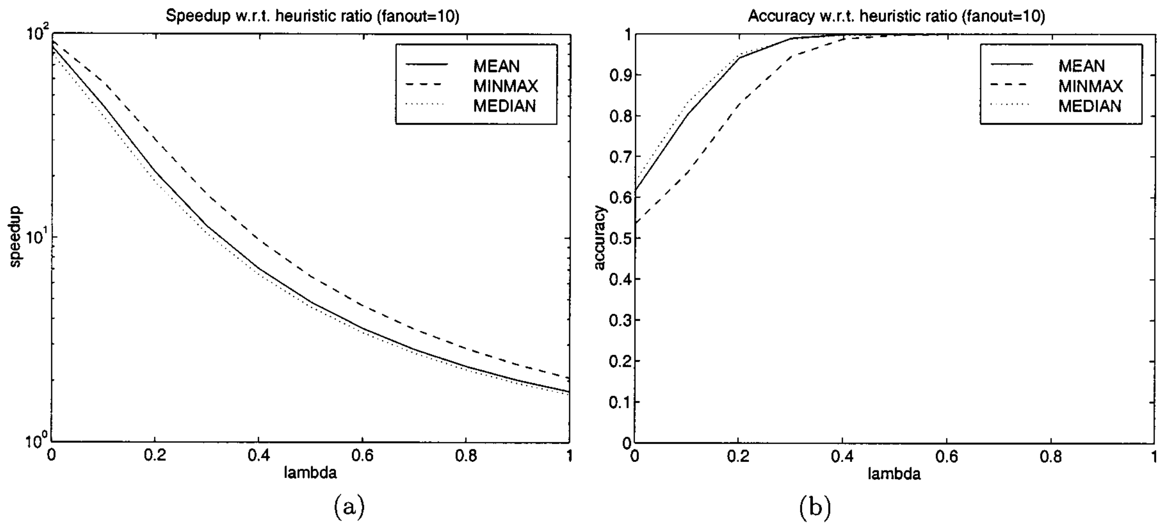


Fig. 9. Speed-up and accuracy of search versus λ. Each plot is for a different cluster centroid. (a) Notice that the minimax centroid gives the best speed-up for exact search at λ = 1. (b) However, the minimax centroid gives uniformly poorer accuracy over a wide range of 0 < λ < 1.

is performed starting at the root of the quadtree and moving to its leaves.

Let s be a node in the quadtree which has been mapped to a node of the pyramid p. The problem is then to find a suitable mapping from the children of the quadtree node, $c(s)$, to the children of the pyramid node, $c(p)$. In general, we would like to choose the mapping that produces the smoothest spatial variations in clusters. To do this we select the mapping that minimizes a total cost function

$$\text{Total Cost} = E_{inter} + E_{intra} + E_{extern} \tag{12}$$

where the three terms represent inter-cluster, intra-cluster, and external costs in node placement. Fig. 6 illustrates the dependencies of terms in the inter-cluster and intra-cluster costs. The inter-cluster terms depend on the similarity of a child node and its neighbors at the coarser scale, while the intra-cluster terms are only between sibling nodes at the same scale. Since there are at most $24 = 4!$ mappings, this optimization can be quickly solved.

In order to precisely define the three cost terms, the position of each node must be specified. The nonnegative integers i_p and j_p denote the position of the pyramid node p on a discrete 2-D unit grid. The four children of the pyramid node p have the (i, j) positions

$$\{(2i_p, 2j_p), (2i_p+1, 2j_p), (2i_p, 2j_p+1), (2i_p+1, 2j_p+1)\}.$$

The inter-cluster and intra-cluster costs are both defined as sums of physical distance divided by dissimilarity. For E_{inter} these terms are between $c(p)$, the children of node p, and $\mathcal{N}(p)$, the four nearest neighbors of p at the same level of the pyramid

$$E_{inter} = \sum_{r \in \mathcal{N}(p)} \sum_{s \in c(p)} n_r n_s \cdot \left(\frac{|2i_r + 0.5 - i_s)| + |2j_r + 0.5 - j_s|}{d(z_r, z_s)} \right).$$

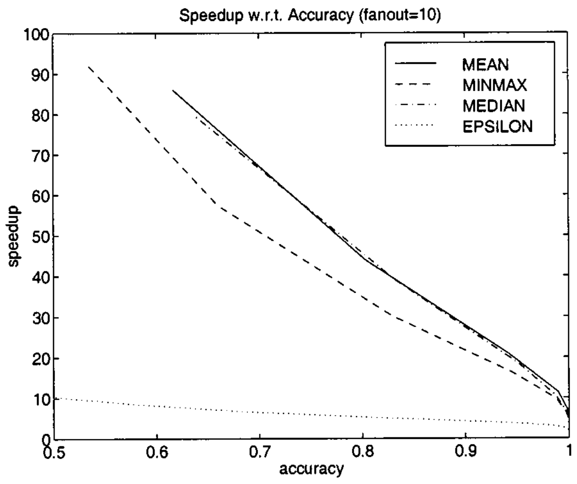


Fig. 10. Speedup versus search accuracy. This plot shows that over a wide range of search accuracy the mean and median centroids are preferable to the minimax.

Here $d(z_r, z_s)$ measures the dissimilarity between the two cluster centroids as defined in Appendix A. Notice that if $\mathcal{N}(p)$ is empty, then this cost term is zero. For E_{intra} the cost terms are computed between elements of $c(p)$

$$E_{intra} = \sum_{r \in c(p)} \sum_{s \in c(p)} n_r n_s \left(\frac{|i_r - i_s| + |j_r - j_s|}{d(z_r, z_s)} \right).$$

The external cost term is used to account for desired attributes of the clusters organization. For example, we choose the following terms where $hue(s)$ and $texture(s)$ are defined in Appendix A and n_p is the number of images in the parent cluster p

$$E_{extern} = \epsilon \sum_{s \in c(p)} n_p^2 \{ i_s\ red(z_s) + j_s\ texture(z_s) \}.$$

The purpose of the external cost term is to break ties when there is more than one mapping that minimizes $E_{inter} + E_{intra}$. Therefore, we choose $\epsilon = 0.0001$ to make the external cost relatively small.

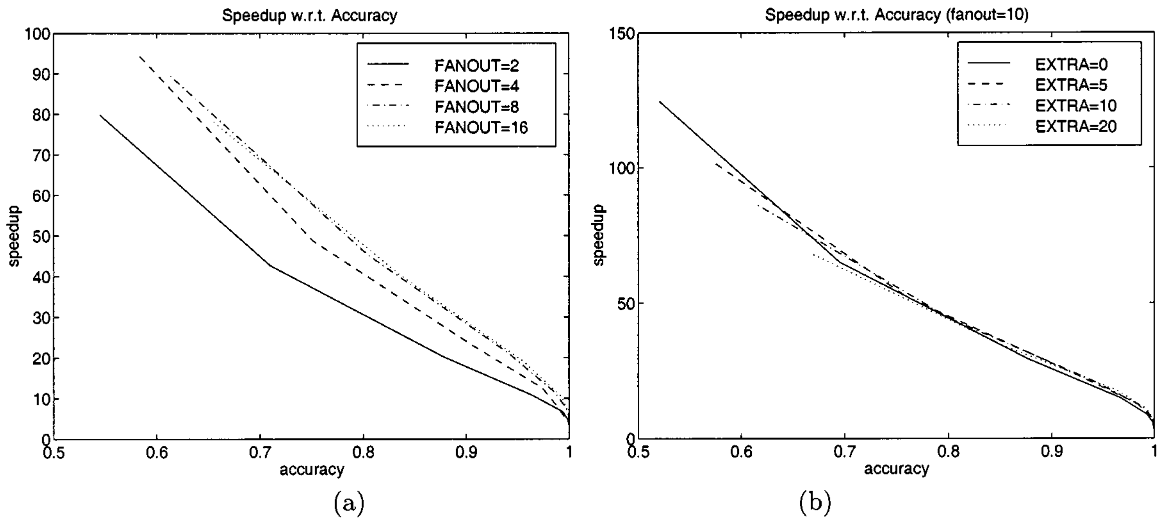


Fig. 11. Speedup versus search accuracy for different tree fan-outs and different numbers of extra returned images. (a) Higher tree fan-outs yields better performance, but requires more computation for tree construction. A fan-out of eight appears sufficient. (b) Speed-up versus search accuracy for different numbers of extra returned images. Returning extra images can improve the speed-up/accuracy tradeoff, but 50% extra returns appears sufficient.

B. Measures of Pyramid Organization

In this section, we introduce a simple intuitive measure of pyramid organization which we call the *dispersion*. The dispersion measures the average distance between similar images in the pyramid. Fig. 7 shows how the distance D_{nm} between images n and m is measured. Notice that $D_{mn} \neq D_{nm}$ because the images may be at different levels of the pyramid. In general, the distance is measured relative to the first image. More formally, the distance is computed as

$$\begin{aligned} D_{nm} =& |i_n + 0.5 - 2^{l(n)-l(m)}(i_m + 0.5)| \\ &+ |j_n + 0.5 - 2^{l(n)-l(m)}(j_m + 0.5)| \end{aligned}$$

where $l(n)$ and $l(m)$ are the levels in the pyramid for images n and m, and (i_n, j_n) and (i_m, j_m) are positions on the 2-D grid. Using this definition, the average distance between image m and its M closest matches may be expressed as

$$\frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)} \tag{13}$$

where $\pi(n, m)$ denotes the index of the mth closest image to image n. Ideally, each image n would be surrounded by its M closest matches as illustrated in Fig. 8. It may be easily shown that for this case the mth closest image is placed at a distance

$$L(m) = \min_k \{k\colon 2k(k+1) \geq m\}$$

so the average distance between n and its M closest matches is

$$\underline{D}(M) = \frac{1}{M} \sum_{m=1}^{M} L(m). \tag{14}$$

Combining the expression of (13) and the lower bound of (14) results in a normalized measure of dispersion

$$Dispersion = \frac{1}{NM\underline{D}(M)} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)}. \tag{15}$$

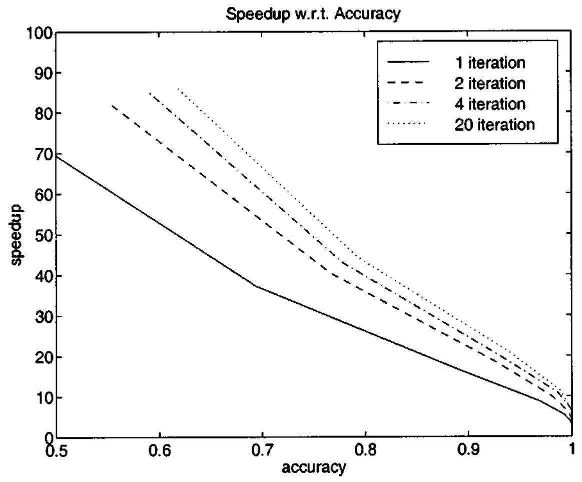


Fig. 12. Speedup versus search accuracy for different number of LBG iteration.

We will use this measure to measure the quality of a similarity pyramid's organization.

C. Incremental Insertions and Deletions

The dynamic nature of image databases often necessitates the insertion of new images and deletion of old images. In the simplest case, images may be inserted or deleted from the sparse distance matrix, and the similarity pyramid may be rebuilt. This method saves some computation by using the existing entries in the sparse distance matrix, but still requires substantial computation for each rebuild of the pyramid. In addition, each insertion or deletion generally can result in substantial discontinuous changes of the pyramid structure, which may not be desirable in particular applications.

Alternatively, images can be more efficiently inserted or deleted by incrementally changing the existing similarity pyramid structure. This can be done by first finding the image in the embedded quadtree that is most similar to the image

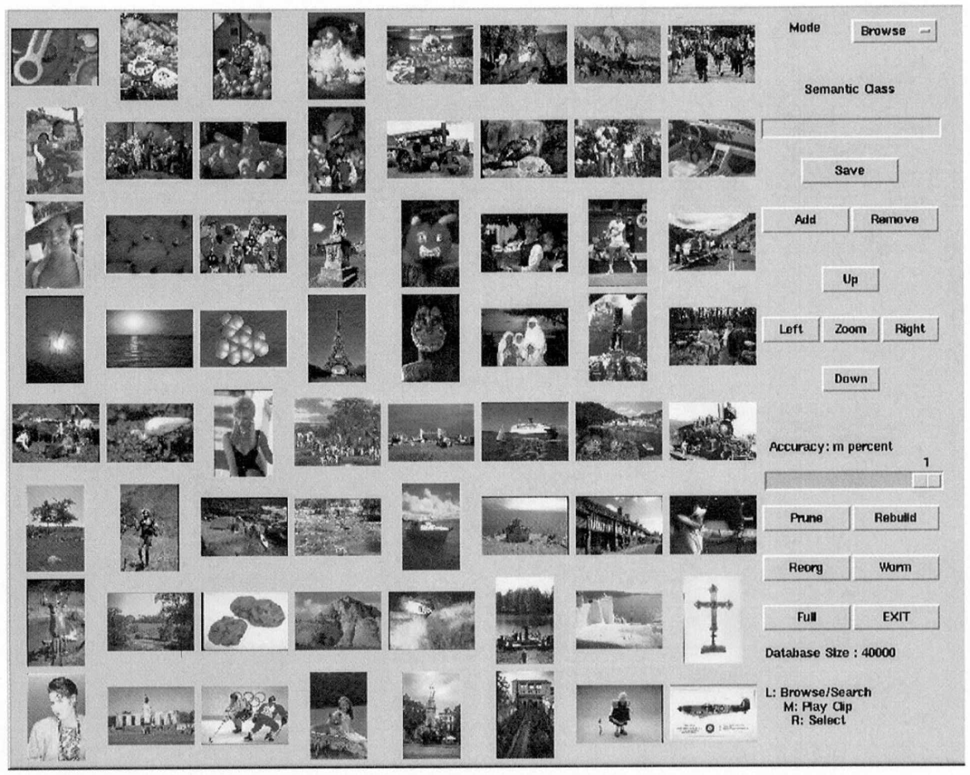


Fig. 13. User environment for viewing a similarity pyramid containing 40 000 images. Each image is an icon for a cluster of images. Buttons allow the user to pan across the database, or move up the pyramid. The user can move down the pyramid by "double clicking" on a specific image icon.

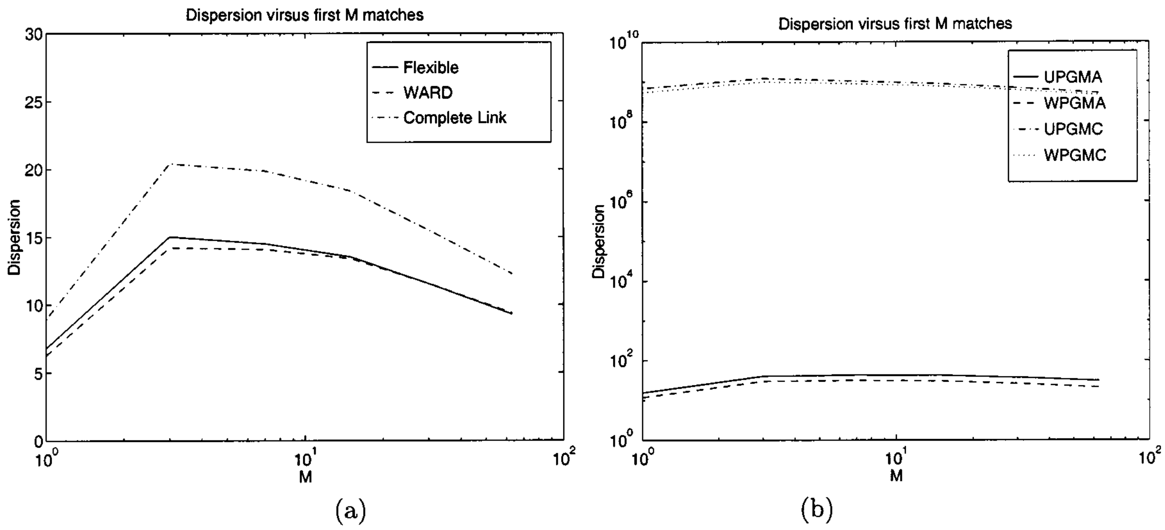


Fig. 14. Dispersion versus M for various bottom-up clustering methods. (a) Flexible, Ward's, and complete link clustering methods. Among these, Ward's and flexible algorithm are the best. (b) UPGMA, WPGMA, UPGMC, and WPGMC clustering methods. All four methods perform poorly.

being inserted. If the parent of this most similar image has less than four children, the new image is added as a sibling of this most similar image. If the parent of this most similar image already has four children, a new internal node is created to replace the most similar image. The new image and most similar image then become children of this new node. This operation is fast, and does not substantially change the existing structure of the pyramid. Deletions are easily implemented by deleting leaves of the embedded quadtree structure.

VI. Experimental Results

In order to test the performance of our search and browsing methods, we use a database of 10 000 natural images. The

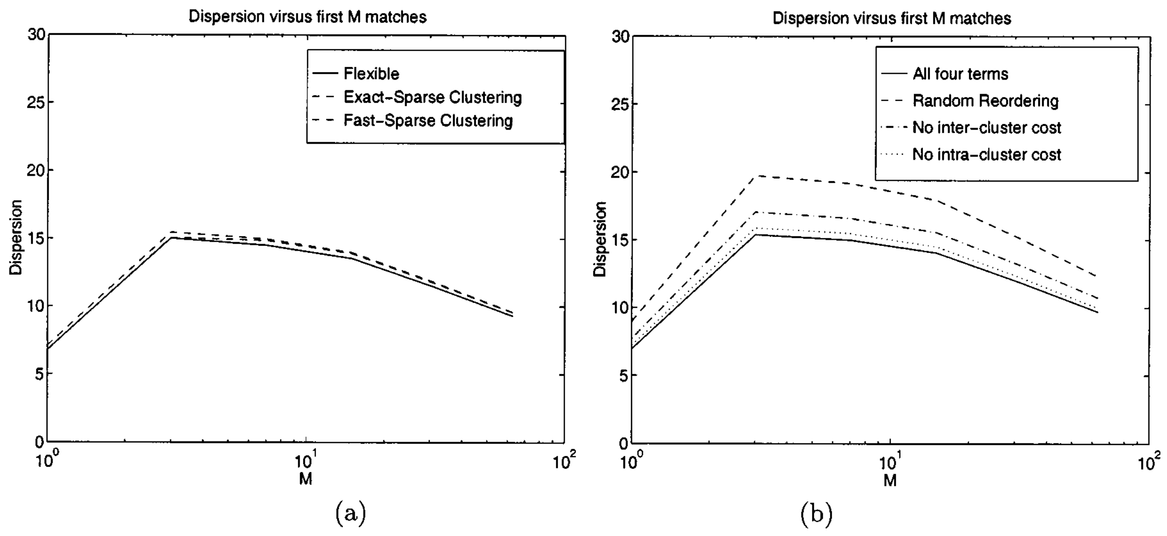


Fig. 15. (a) Comparison of flexible clustering with full and sparse matrices and fast-sparse method. All three variations of flexible clustering seems to perform similarly. Effect of reordering algorithm. (b) The performance of reordering algorithm using (12) is much better than using random reordering. It also shows that the effect of inter-cluster cost E_{inter} and intra-cluster cost E_{intra}.

database contains a variety of images with varying color, texture and content. Appendix A describes the 211 element feature vector that was extracted for each image [33]. In all cases, the L_1 norm was used as the distance metric.

Section VI-A presents results of search experiments and experimentally determines the tradeoffs between search accuracy and speed. Section VI-B presents objective measures of similarity pyramid organization, and shows how these measures vary with methods for constructing the pyramid. We also apply the fast search algorithms of Section IV to the problem of pyramid design, and measure the computational savings of this approach.

A. Search Experiments

In this section, all trees were designed using a coarse-to-fine procedure as described in Section III-A. Unless otherwise stated, a fan-out of 10 was used $(K = 10)$, centroids were computed using feature means as in (1), and 20 iterations of the K-means algorithm were performed using initial code words that were randomly selected from the corresponding group of image feature vectors.

Unless otherwise stated, we selected the ten best image matches after searching for 20 images (i.e., we searched for ten extra images.) Each plot was then formed by averaging results over a set of 1000 randomly selected query images representing approximately 10% of the database. Computational speed-up is defined as the ratio of the total number of images to the number of tree nodes search. More specifically,

$$\text{speed-up} = \frac{\#\text{ of images } N}{\text{NodesSearched}}$$

where NodesSearched is defined in the algorithm of Fig. 4. We note that since the total number of tree nodes is $2N - 1$, the speed-up can be less than one, but this was never observed.

λ selection: Fig. 9(a) and (b) show the speed-up and accuracy as a function of the free parameter λ. The three plots illustrate the results using mean, median and minimax centroids. It is interesting to note that for λ = 1 (exact search) the minimax centroid gives better speed-up than the mean and median [19]. However, the speed-up for exact search is limited to approximately 2 which is much less than can be achieved with approximate search.

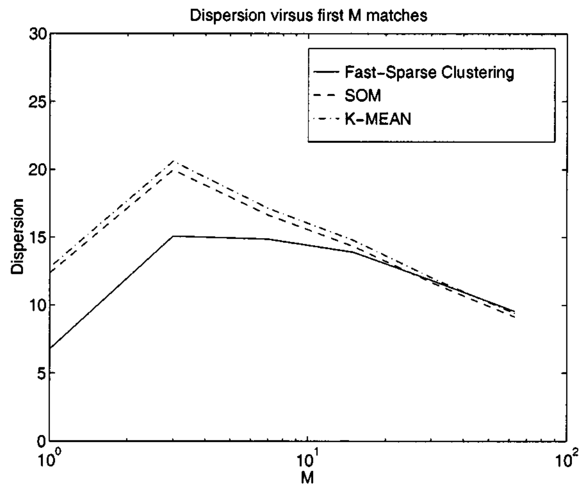


Fig. 16. Dispersion versus M for K-means and SOM trees and fast-sparse flexible. Apparently, SOM outperforms K-means everywhere. The flexible method has better performance for low values of M which seem to be most important in practice.

However, Fig. 10 shows the direct tradeoff of speed-up versus accuracy. In this plot it is clear that for even very high accuracy levels the mean and median centroids substantially out perform the minimax centroid. In practice, accuracies greater than 90 percent are probably not necessary for this application. This indicates that speed-ups of 20–40 are possible with good to excellent accuracy. Based on these results, the mean centroid seems preferable since it is much more efficient to compute than the median centroid. Fig. 10 also compares the fast search algorithm to a standard approximate search method known as $(1+\epsilon)$

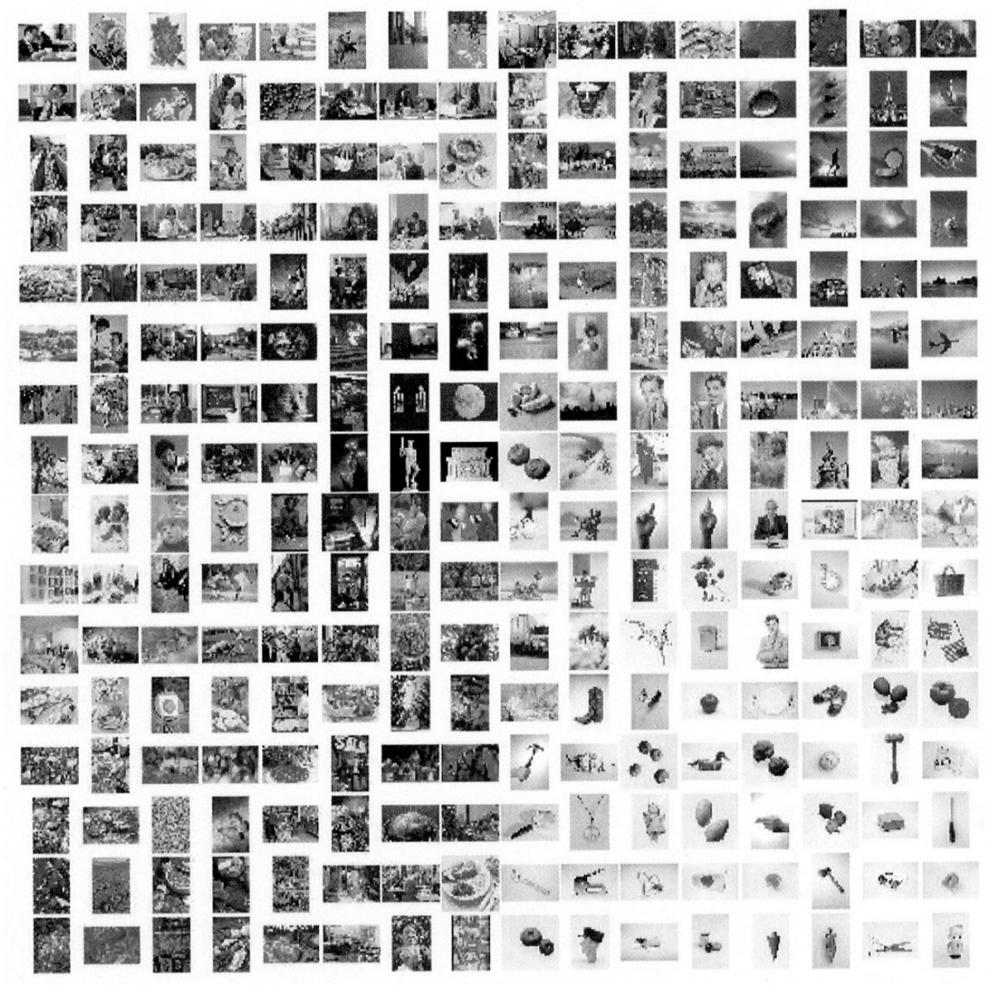


Fig. 17. Typical 16 × 16 layout at level 4 for fast sparse flexible algorithm with dispersion slightly higher than average.

search [32]. The $(1+\epsilon)$ approximate search gives poor performance in this application because it does not exploit the specific structure of the problem.

Effects of fan-out and extra returns: Fig. 11(a) shows the search performance for a variety of different fan-out rates in the tree. A tree with low fan-out requires less computation to construct since the computation of the LBG algorithm is of order $N(\text{fan-out})(\text{tree depth})$. However, Fig. 11(a) indicates that a fan-out less than eight can substantially reduce performance.

Fig. 11(b) shows that searching for extra images can improve performance. However, the effect is rather weak. This is because returning extra images substantially increases the computation, thereby reducing the accuracy/speed-up tradeoff. Fig. 11(b) indicates that returning five extra images is sufficient to gain most of the benefit.

LBG iteration: Fig. 12 shows that LBG iteration can improve performance. However, the effect is rather weak after four iterations. Furthermore, the larger the number of return images, the smaller the improvement of multiple LBG iterations.

B. Browsing Experiments

In this section, we use the dispersion measure to compare various methods for constructing similarity pyramids. In each case, we average performance over 20 runs; and for the fast-sparse clustering algorithm, we use a matrix sparsity of 1%, a search parameter of $\lambda = 0.1$, and a top-down tree fanout of $K = 10$ constructed with $I = 2$ K-means iterations.

Fig. 13 shows the user environment for viewing level $l = 3$ of a similarity pyramid containing over 40 000 images. Each image icon represents a cluster of many images. The buttons in the middle right hand region (labeled up, left, right, down, and zoom) allow the user to pan across the database, or move up the pyramid. The user can move down the pyramid by "double clicking" on a specific image icon.

In Fig. 14, we compare the quality of a variety of bottom-up clustering methods. The Ward's and flexible clustering method with $\beta = -1$ work the best among these methods, producing the smallest dispersion. The four methods UPGMA, WPGMA, UPGMC, and WPGMC all perform very poorly. Intuitively, this is because these conserving algorithms tend to generate very deep trees. This means that on average images will be dispersed far away.

Fig. 15(a) compares our fast-sparse clustering to exact-sparse clustering (i.e. fast-sparse clustering with $\lambda = 1$) and standard flexible clustering. Notice that the fast-sparse clustering gives essentially the same performance as the much more computationally and memory intensive algorithms. The fast-sparse clustering requires 1% of the memory of the standard flexible clustering algorithm, and it requires 6.6% of the computation of the exact sparse clustering algorithm to compute the proximity matrix.

Fig. 15(b) illustrates the importance of top-down reordering on a similarity pyramid built with fast-sparse clustering. Random reordering yields the poorest performance. It is also interesting to note that the inter-cluster cost, E_{inter}, is more important then the intra-cluster cost, E_{intra}. In Fig. 16, we compare fast-sparse clustering to top-down K-means clustering, and the SOM-based clustering described in Appendix B. Notice that fast-sparse clustering has significantly better performance than SOM over a wide range of M. In practice, we have found that the dispersion is most important for small values of M. Intuitively, images which are "miss-placed" in the database tend to increase the dispersion for these small values of M. Such "miss-placed" images can be very difficult to locate making them effectively lost to a user. Both the results of K-means and SOM are averaged over 20 different runs. Not surprisingly, SOM outperforms K-means since the SOM method uses $K = 64$ clusters rather than the $K = 4$ clusters of the K-means algorithm. However, SOM also requires roughly 16 times the computation of K-means.

In Fig. 17, we show the 16 × 16 level similarity pyramid built using fast-sparse clustering. Notice how the thumbnail images are placed so that images with similar color and texture are spatially adjacent.

VII. Conclusion

We have shown that hierarchical trees and pyramids are very effective for both searching and browsing large databases of images. Generally, top-down tree growing strategies seem to be better suited to fast search, while bottom-up tree growing strategies seem better for browsing.

We proposed a best-first implementation of branch and bound search which allows us to efficiently search for the M closest images to any query image. Our fast search algorithm can be used for both exact and approximate search, but we found that approximate search yeilds a much better speed/accuracy tradeoff for our applications. In fact, even at accuracies of 90%, the approximate search algorithm reduced computation by a factor greater than 25, while exact search only resulted in a speed-up of two.

We proposed a data structure called a similarity pyramid for browsing large image databases, and we proposed a fast-sparse clustering method for building these pyramids efficiently. The fast-sparse clustering method is based on the flexible agglomerative clustering algorithm, but dramatically reduces memory use and computation by using only a sparse proximity matrix and exploiting our approximate branch and bound search algorithm. We found that the method for mapping the clustering to a pyramid can make a substantial difference in the quality of organization. Finally, we proposed a dispersion metric for objectively measuring pyramid organization, and we found that the dispersion metric correlated well with our subjective evaluations of pyramid organization.

Appendix
Similarity Measures for Images

All images in the database consisted of either 96 × 64 or 64 × 96 thumbnails rescale from the original images. Based on our previous work [33], the feature vector x_i for image i includes the global color, texture, and edge histograms. The dissimilarity function $d(q, x)$ is defined as the L_1 norm between corresponding histograms of two images.

The color feature were formed by independently histograming the three components C_L, C_a and C_b of the CIELa^*b^* color space. We chose the number of histogram bins so that the resolution of each bin was approximately $6\Delta E$. In addition, we smoothed the histogram by applying a Gaussian filter kernel with a standard deviation of $6\Delta E$. The texture feature was formed by histograming the magnitude of the local image gradient for each color component. More specifically, D_xL and D_yL are defined as the x and y derivatives of the L component computed using the conventional Sobel operators. Then $T_L = \sqrt{(D_xL)^2 + (D_yL)^2}$, and T_a, T_b are defined similarly. We chose the number of histogram bins so that the resolution of each bin was also approximately $6\Delta E$. The edge feature was formed by thresholding the edge gradient and then computing the angle of the edge for all points that exceed the threshold

$$\Theta_L = \begin{cases} \arctan(D_xL, D_yL), & T_L \geq \sigma_L \\ \emptyset, & T_L < \sigma_L. \end{cases}$$

The threshold σ_L was computed as the standard deviation of the L component for the particular image. The values of Θ_a and Θ_b are computed similarly. We chose the number of histogram bins so that the resolution of each bin was also approximately $\pi/8$. An extra bin is reserved for the case of $\Theta = \varnothing$.

With these definitions, we also define some extended features which can be approximately derived from the histogramed feature vector

$$\begin{aligned} red(x_i) &\triangleq \overline{C_a} \\ blue(x_i) &\triangleq \overline{C_b} \\ texture(x_i) &\triangleq \frac{(\overline{T_L} + \overline{T_a} + \overline{T_b})}{3}. \end{aligned}$$

VIII. SOM Pyramid Construction

We use SOM [34] to create an 8 × 8 grid of clusters that forms the pyramid at level $l = 3$. This keeps the computation manageable for large databases. As with K-means, conventional SOM will create a tree which is too unbalanced, so we apply the constraint that each cluster must contain $\lceil N/64 \rceil$ or fewer elements. The remaining levels of the pyramid were grown with the K-means algorithm. For all our experiments, we used 20 iterations of SOM with 5 iterations applied each at neighborhood sizes of 7 × 7, 5 × 5, 3 × 3, and 1 × 1.

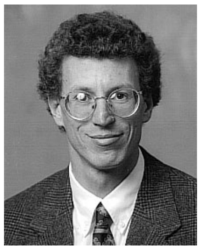
References

[1] M. Myron Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image content: The QBIC system," *IEEE Computer*, pp. 23–31, Sept. 1995.

[2] J. MacCuish, A. McPherson, J. Barros, and P. Kelly, "Interactive layout mechanisms for image database retrieval," in *Proc. SPIE/IS&T Conf. Visual Data Exploration Analysis III*, vol. 2656, San Jose, CA, Jan. 31–Feb. 2, 1996, pp. 104–115.
[3] Y. Rubner, L. Guibas, and C. Tomasi, "The earth mover's distance, multi-dimensional scaling, and color-based image retrieval," in *Proc. ARPA Image Understanding Workshop*, May 1997.
[4] M. M. Yeung and B. Liu, "Efficient matching and clustering of video shots," in *Proc. IEEE Int. Conf. Image Processing*, vol. I, Washington, DC, Oct. 26, 1995, pp. 338–341.
[5] M. M. Yeung and B.-L. Yeo, "Video visualization for compact presentation and fast, browsing of pictorial content," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 771–785, Oct. 1997.
[6] R. Milanese, D. Squire, and T. Pun, "Correspondence analysis and hierarchical indexing for content-based image retrieval," in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, Lausanne, Switzerland, Sept. 19, 1996, pp. 859–862.
[7] H. J. Zhang and D. Zhong, "A scheme for visual feature based image indexing," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases III*, vol. 2420, San Jose, CA, Feb. 10, 1995, pp. 36–46.
[8] M. T. Orchard, "A fast nearest-neighbor search algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Processing*, Toronto, Ont., Canada, May 1991, pp. 2297–2300.
[9] W. A. Burkhard and R. M. Keller, "Some approaches to beat-match file searching," *Commun. ACM*, vol. 16, no. 4, pp. 230–236, Apr. 1973.
[10] V. Ruiz, "An algorithm for finding nearest neighbors in (approximately) constant time," *Pattern Recognit. Lett.*, vol. 4, pp. 145–157, July 1986.
[11] J. Barros, J. French, W. Martin, P. Kelly, and M. Cannon, "Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases IV*, vol. 3022, San Jose, CA, Feb. 2, 1996, pp. 392–403.
[12] A. Berman and L. Shapiro, "Efficient image retrieval with multiple distance measures," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases V*, vol. 3022, San Jose, CA, Feb. 14, 1997.
[13] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, pp. 579–589, 1991.
[14] D. A. White and R. Jain, "Similarity indexing: Algorithms and performance," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases IV*, vol. 2670, San Jose, CA, Feb. 2, 1996, pp. 62–73.
[15] R. Ng and A. Sedighian, "Evaluating multi-dimensional indexing structure for images transformed by principal component analysis," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases IV*, vol. 2670, San Jose, CA, Feb. 2, 1996, pp. 50–61.
[16] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comput.*, pp. 750–653, July 1975.
[17] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proc. ACM SIGMOD Int. Conf. Management Data*, San Jose, CA, June 1995, pp. 71–79.
[18] D. A. White and R. Jain, "Similarity indexing with the S-tree," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases IV*, vol. 2670, San Jose, CA, Feb. 2, 1996, pp. 62–73.
[19] 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 *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases V*, vol. 3022, San Jose, CA, Feb. 14, 1997, pp. 110–120.
[20] J.-Y. Chen, C. A. Bouman, and J. P. Allebach, "Fast image database earch using tree-structured vq," in *Proc. IEEE Int. Conf. Image Processing*, vol. 2, Santa Barbara, CA, Oct. 29, 1997, pp. 827–830.
[21] J.-Y. Chen, C. A. Bouman, and J. Dalton, "Similarity pyramids for browsing and organization of large image databases," in *Proc. SPIE/IS&I Conf. Human Vision Electronic Imaging III*, vol. 3299, San Jose, CA, Jan. 29, 1998.
[22] J.-Y. Chen, C. A. Bouman, and J. Dalton, "Active browsing using similarity pyramids," in *Proc. 36th Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Nov. 4, 1998.
[23] ——, "Active browsing using similarity pyramids," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases VII*, vol. 3656, San Jose, CA, Jan. 29, 1999, pp. 144–154.
[24] G. N. Lance and W. T. Williams, "A general theory of classificatory sorting strategies. I. Hierarchical systems," *Comput. J.*, vol. 9, pp. 373–380, 1966.
[25] P. H. A. Sneath and R. R. Sokal, *Numerical Taxonomy*, P. H. A. Sneath and R. R. Sokal, Eds. San Francisco, CA: Freeman, 1973.
[26] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical, Statistics, Probability*, 1964, pp. 281–297.
[27] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, Jan. 1980.
[28] M. M. Yeung, B.-L. Yeo, and B. Liu, "Extracting story units from long programs for video browsing and navigation," in *IEEE Int. Conf. Multimedia Computing Syst.*, Hiroshima, Japan, June 21, 1996, pp. 296–305.
[29] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, A. K. Jain and R. C. Dubes, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[30] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.
[31] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Eds. New York: McGraw-Hill, 1990.
[32] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," in *Proc. 5th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1994, pp. 573–582.
[33] J.-Y. Chen, C. A. Bouman, and J. P. Allebach, "Multiscale branch and bound image database search," in *Proc. SPIE/IS&T Conf. Storage Retrieval Image Video Databases V*, vol. 3022, San Jose, CA, Feb. 14, 1997, pp. 13–14.
[34] T. Kohonen, *Self-Organizing Maps*, T. Kohonen, Ed. Berlin, Germany: Springer, 1995.

Jau-Yuen Chen was born in Pingtong, Taiwan, R.O.C., on January 25, 1963. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, in 1985 and 1987, respectively. He received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 1999.

From 1987 to 1994, he was an Assistant Scientist at the Chung-Shan Institute of Science and Technology, Taiwan. He is currently with Epson, Palo Alto, CA.



Charles A. Bouman (S'86–M'89–SM'97) received the B.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1981 and the M.S. degree from the University of California, Berkeley, in 1982. In 1989, he received the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, under the support of an IBM graduate fellowship.

From 1982 to 1985, he was a Full Staff Member at Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, In 1989, he joined the faculty of Purdue University, West Lafayette, IN, where he is an Associate Professor in the School of Electrical and Computer Engineering. His research focuses on the use of statistical image models, multiscale techniques, and fast algorithms in applications, such as multiscale image segmentation, fast image search and browsing, and tomographic image reconstruction.

Dr. Bouman is a member of the SPIE and IS&T professional societies. He has been an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING and is currently an associate editor for the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. He was a member of the ICIP 1998 organizing committee, and is currently a chair for the SPIE Conference on Storage and Retrieval for Image and Video Databases and a member of the IEEE Image and Multidimensional Signal Processing Technical Committee.

John C. Dalton received the B.E.E. and M.E.E. degrees from the University of Delaware, Newark, in 1981 and 1983, respectively.

He has been a Senior Research Scientist at Tektronix, Apple Computer, and Ricoh. His research interests have included color hardcopy, electronic publishing, image processing, human visual models, adaptive learning systems, computational chemistry, and content-based retrieval, and visualization of multimedia databases. He was one of the founders of OSC Software, which pioneered multitrack digital audio software for personal computers. Currently, he is a Principal at Synthetik Software, San Francisco, CA.