Similarity Pyramids for Browsing and Organization of Large Image Databases [*†]

Jau-Yuen Chen[†], Charles A. Bouman[†] and John C. Dalton[‡]

[†]School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285
{jauyuen,bouman}@ecn.purdue.edu

[‡]Ricoh California Research Center
johnd@synthetik.com

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 presents 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. We then present a objective measure of pyramid organization called dispersion, and we use it to show that our fast-sparse cluster method produces better similarity pyramids than top down approaches.

1. 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 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 a general, a browsing environment seeks to logically and predictably organize the database so that a user 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, Guibas, and Tomasi[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.

[*]This work was supported by Apple Computer.

[†]Appeared at Proc. of SPIE/IS&T Conf. on *Human Vision and Electronic Imaging III*, vol. 3299, pp. 563-575, San Jose, CA, Jan. 26-29, 1998.

Yeung, Yeo and Liu 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.

Zhang and Zhong[6] proposed a hierarchical self-organizing map (HSOM) which used the SOM algorithm to organize a complete database of images into a 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 this paper, we present a hierarchical browsing environment which we call a similarity pyramid that efficiently organizes databases so that similar images are located nearby each other. The key to our method is the use of tree structured database organization. We will concentrate our attention on bottom-up or agglomerative clustering methods for constructing these trees because these bottom-up methods tend to yield better results. However, 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 presented in[7] to dramatically reduce both memory and computation requirements.

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 two dimensional 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. Finally, we propose a quality measure for the pyramid organization which we call *dispersion*, and use this measure to evaluate various approaches to pyramid design.

2. 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 dis-similarity 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 our previous research,[8] and we will use the L_1 norm as a measure of feature distance.

We will use tree structures to hierarchically organize images into similar groups, thereby allowing users 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 $C_i = \{i\}$.

3. CLUSTERING FOR PYRAMID DESIGN

In this section, we will introduce a fast-sparse clustering algorithm which implements the standard flexible agglomerative (bottom-up) clustering algorithm,[9] but with dramatically reduced memory and computational requirements. We will then use the fast-sparse clustering algorithm to design the image browsing environments of Section 4.

3.1. Agglomerative Clustering Algorithms

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.[10,11] 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 2.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.[9]

$$d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} - \gamma |d_{hi} - d_{hj}| \tag{1}$$

Here α_i, α_j, β and γ are coefficients which depend on some property of the clusters. Most standard clustering algorithms result from the proper choice of these coefficients.[10,11]

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 "non-conformist" 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[9] which uses the update rule

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

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,

$$d_{hk} = d_{hi} + d_{hj} - d_{ij} \ . \tag{3}$$

3.2. 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 are then computed using the fast search algorithm of Chen, Bouman and Allebach.[7]

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 (3) 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 (3) 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

$$d_{hk} = \begin{cases} d_{hi} + d_{hj} - d_{ij} & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are available} \\ \hat{d}_{hi} + d_{hj} - d_{ij} & \text{if } d_{hi} \text{ is missing} \\ d_{hi} + \hat{d}_{hj} - d_{ij} & \text{if } d_{hj} \text{ is missing} \\ \text{Do not compute} & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are missing} \end{cases} \ . \tag{4}$$

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

$$\tilde{d}_{hk} = \tilde{d}_{hi} + \tilde{d}_{hj} \tag{5}$$

in place of the recursion of (3). 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.

$$\tilde{d}_{hk} = \sum_{m \in C_h} \sum_{n \in C_k} d(x_m, x_n) \tag{6}$$

We may use (6) to approximate the missing term $\hat{d}_{hi}$ of (4). We do this in terms of the quantity r_i, the distance between image i and its M^{th} closest match. More specifically, let $\pi(i,j)$ be the index of the j^{th} closest match to image i. Then the distance to the M^{th} closest match is

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

The required lower bound is then given by

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

```
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 $d_{i\,j} \leftarrow d(x_i, x_j)$
    }
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 (4) with $\hat{d}_{hi} = \max(n_h r_h \, , \, n_i r_i)$
    }
```

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

$$\geq \max\left(\sum_{m \in C_h} \sum_{n \in C_i} r_m \, , \, \sum_{m \in C_h} \sum_{n \in C_i} r_n \right)$$

$$= \max\left(n_i \sum_{m \in C_h} r_m \, , \, n_h \sum_{n \in C_i} r_n \right)$$

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

$$\hat{d}_{hi} \stackrel{\triangle}{=} \max\left(n_i \sum_{m \in C_h} r_m \, , \, n_h \sum_{n \in C_i} r_n \right) \tag{7}$$

were the summation terms may be recursively computed as part of the clustering algorithm. Figure 1 shows the complete sparse clustering algorithm where S_i is the set of sparse entries of each image i, and the set $S \times S_i = \{(i,j) : i \in S \text{ and } j \in S_i\}$.

In previous research,[7] we proposed a fast search algorithm for computing the M closest matches to an image. This algorithm can reduce search computation by a factor of 25 while retaining a search accuracy of 90% on our sample database of 10,000 images. We will use this fast search algorithm to efficiently compute the entries of the sparse matrix. We note that our fast search algorithm 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.

3.3. Binary to quad-tree 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

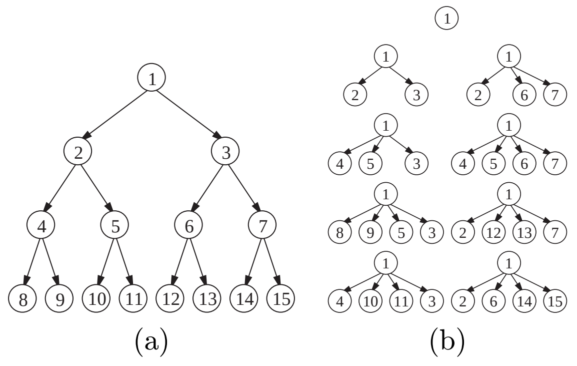


Figure 2. All 4 partitions of a binary tree node $s = 1$. (a) Original binary tree with root node $s = 1$. (b) The 9 unique partitions of the node $s = 1$. Each partition contains 4 or less nodes that partition the original node.

a manner that minimizes a cost criteria. We are particularly interested in the quad-tree case (K=4) since this will be important for the browsing application of Section 4.

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

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

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

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

Figure 2 shows all 4 partitions of a binary tree node labeled as $s = 1$. Each of the nine possible partitionings contains 4 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

$$P^* = \min_{P \in \mathbf{P}(s,K)} \left\{ \max_{j \in P} \ n_j \right\} \qquad (8)$$

where $n_j = |C_j|$.

4. DATABASE BROWSING

In this section, we describe the similarity pyramid data structure that allows users to move through the database in a natural manner. The similarity pyramid is created by mapping each level of a quad-tree onto an 2-D grid to form a level of the pyramid. The pyramid differs from the quad-tree 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

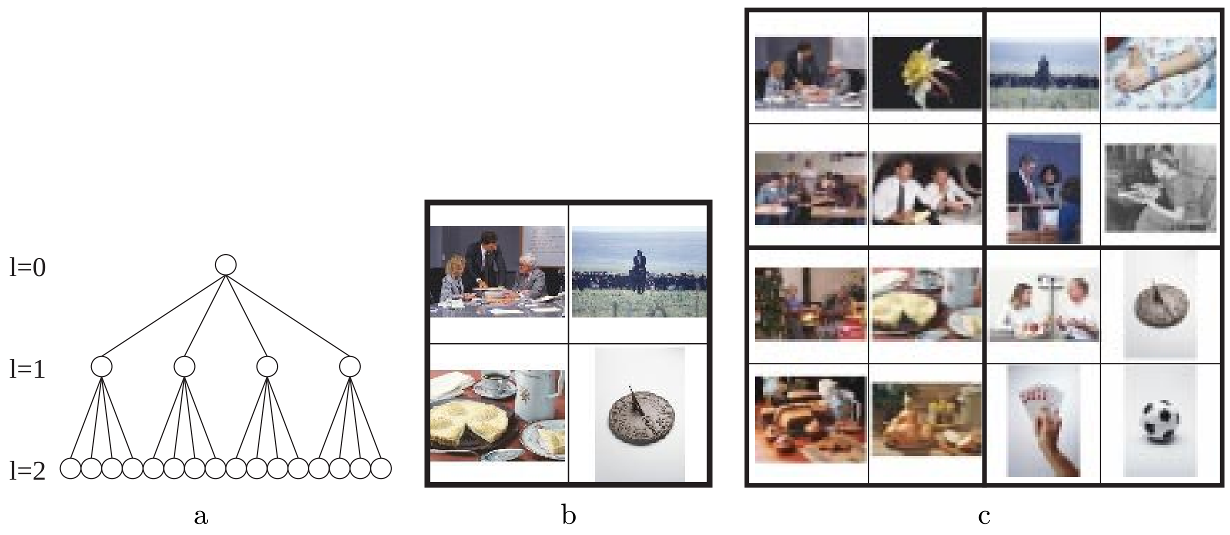


Figure 3. This figure illustrates how nodes are mapped from the quad-tree to the similarity pyramid. There are 24 possible mappings of 4 children nodes to four pyramid nodes. The mapping is chosen to maximize spatial smoothness in image characteristics.

spatial registration. This allows users to move up and down through levels of the pyramid while retaining relative orientation in the database.

Figure 3 illustrates a simple example of a similarity pyramid and its associated quad-tree. Figure 3(a) shows three levels of a quad-tree corresponding to $l = 0$, 1 and 2. Each node represents a cluster of images in the database with the leaf nodes representing individual images. Figure 3(b) shows level $l = 1$ of the similarity pyramid while Fig. 3(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 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 me

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

Figure 7 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 4 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.

4.1. Quad-Tree to Pyramid Mapping

In this section, we describe our method for mapping nodes of the quad-tree to nodes of the similarity pyramid. This mapping is performed starting at the root of the quad-tree and moving to its leaves.

Let s be a node in the quad-tree 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 quad-tree node, $c(s)$, to the children of the

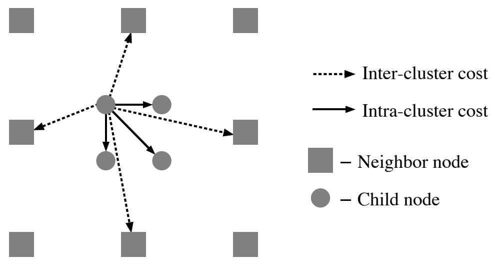


Figure 4. This figure illustrates the inter-cluster and intra-cluster cost terms used to organize images in the pyramid. The neighbor nodes are at level l of the pyramid, while the children nodes are at level $l-1$.

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

$$\text{Total Cost} = E_{inter} + E_{intra} + E_{extern} \tag{9}$$

where the three terms represent inter-cluster, intra-cluster, and external costs in node placement. Figure 4 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 non-negative 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

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

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.

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

Here $d(z_r, z_s)$ measures the dissimilarity between the two cluster centroids. 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 measures of image hue and texture, and n_p is the number of images in the parent cluster p.

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

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.

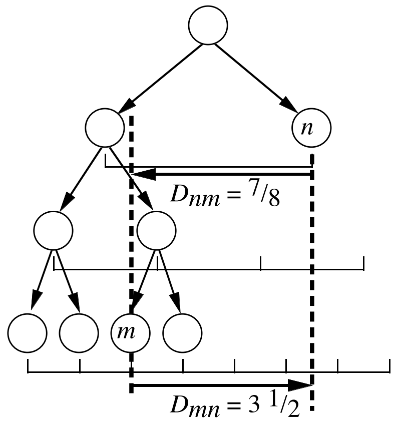


Figure 5. This figure illustrates the distance measured between two images in the similarity pyramid. The distance is an asymmetric function because it is measured relative to the first image.

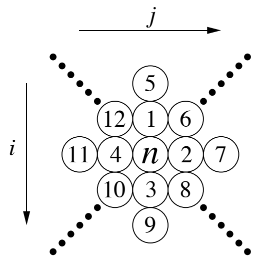


Figure 6. An example of a dense packing of the M closest matches to image n. The M images are arranged in a diamond shape such that the physical distance $D_{n\pi(n,m)}$ is nondecreasing with respect to m.

4.2. 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. Figure 5 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

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

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

$$\frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)} \tag{10}$$

where $\pi(n,m)$ denotes the index of the m^{th} closest image to image n. Ideally, each image n would be surrounded by its M closest matches as illustrated in Fig. 6. It may be easily shown that for this case the m^{th} closest image is placed at a distance

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

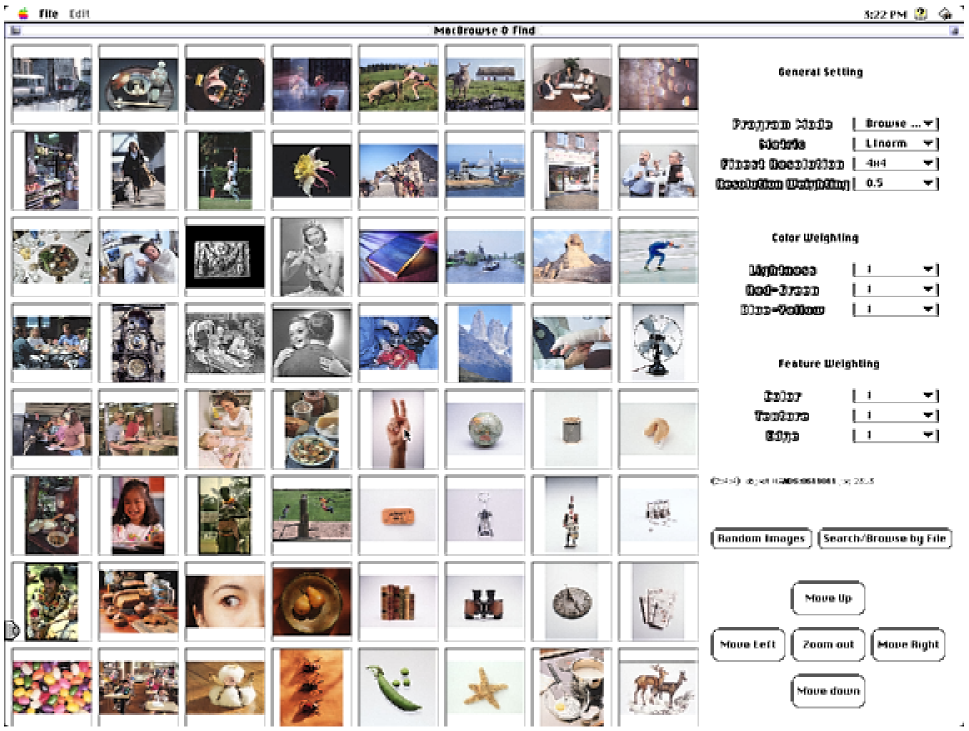
so the average distance between n and its M closest matches is

$$\underline{D}(M) = \frac{1}{M} \sum_{m=1}^{M} L(m) \tag{11}$$

Combining the expression of (10) and the lower bound of (11) results in a normalized measure of dispersion.

$$Dispersion = \frac{1}{NM\underline{D}(M)} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)} \tag{12}$$

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



Program Mode: Browse ...
Metric: Linorm
Finest Resolution: 4x4
Resolution Weighting: 0.5
Lightness: 1
Red-Green: 1
Blue-Yellow: 1
Color: 1
Texture: 1
Edge: 1

Figure 7. This figure shows the user environment for viewing a similarity pyramid containing 10,000 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.

5. 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 $\lambda = 0.1$ as described in.[7]

In order to test the performance of our search and browsing methods, we use a database of 10,000 natural images. The database contains a variety of images with varying color, texture and content. We will use the image similarity metric described in our previous work.[8] This metric uses a 211 element feature vector together with an L_1 norm (i.e. histogram intersection).

Figure 7 shows the user environment for viewing level $l = 3$ of a similarity pyramid containing over 10,000 images. Each image icon represents a cluster of many images. The buttons in the lower right hand region 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 Figure 8, 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.

Figure 9(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

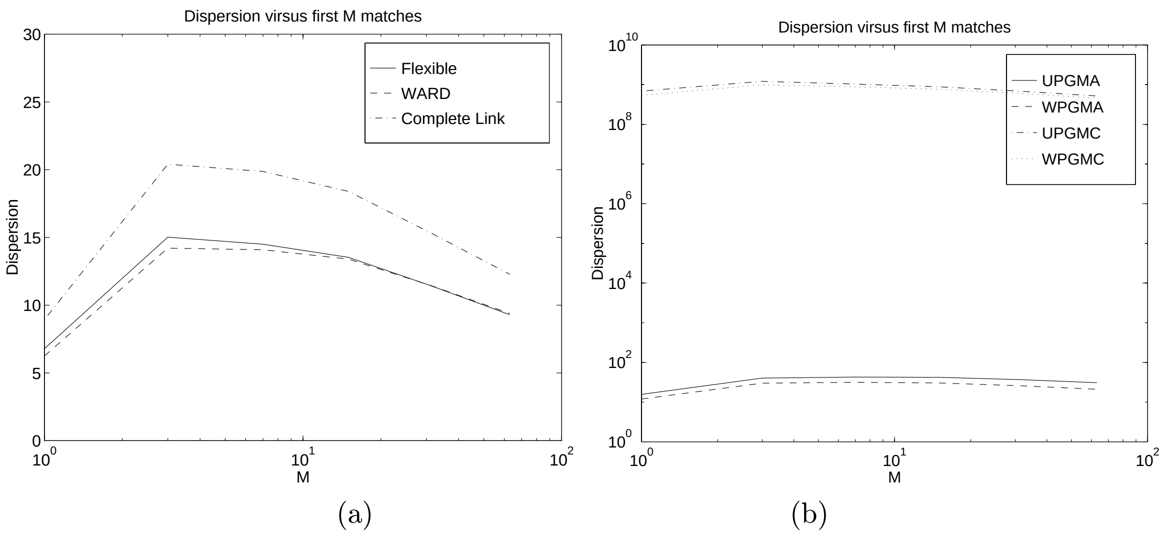


Figure 8. Dispersion versus M for various bottom-up clustering methods. (a) Flexible, Ward's and complete link clustering methods. Among these, Ward's and flexible algorithm are the best. (b) UPGMA, WPGMA, UPGMC, and WPGMC clustering methods. All four methods perform poorly.

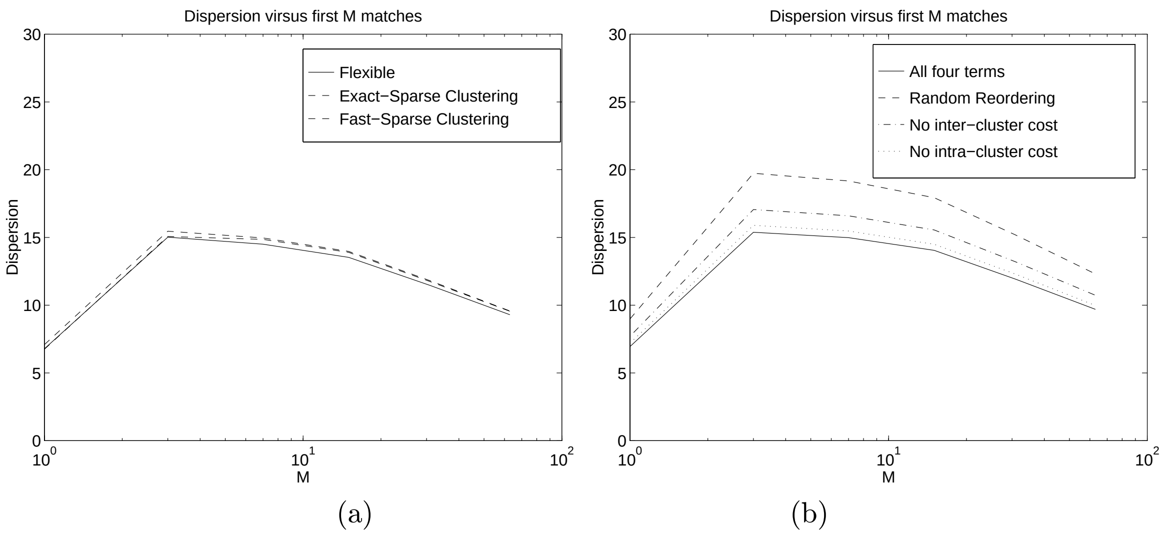


Figure 9. (a) Comparison of flexible clustering with full and sparse matrices and fast-sparse method. All three variations of flexible clustering seems to perform similarly. Effect of reordering algorithm. (b) The performance of reordering algorithm using equation 9 is much better than using random reordering. It also shows that the effect of inter-cluster cost, E_{inter} and intra-cluster cost, E_{intra}.

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.

Figure 9(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 figure 10, we compare fast-sparse clustering to top-down K-means clustering, and the SOM-based clustering. 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.

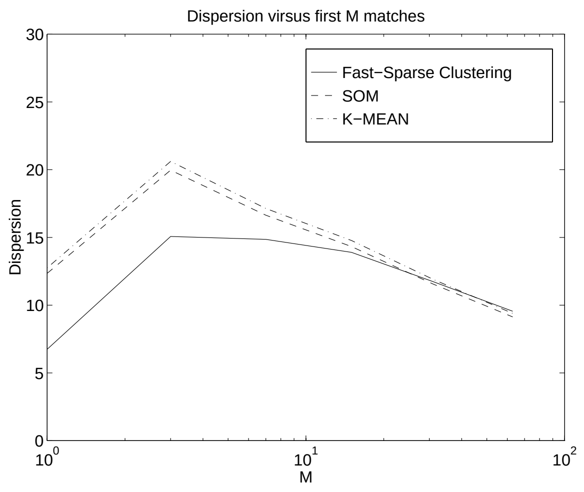


Figure 10. Dispersion versus M for K-means and SOM trees and fast-sparse flexible. Apparently, SOM outperforms K-means everywhere. The flexible method has better performance for low values of M which seem to be most important in practice.

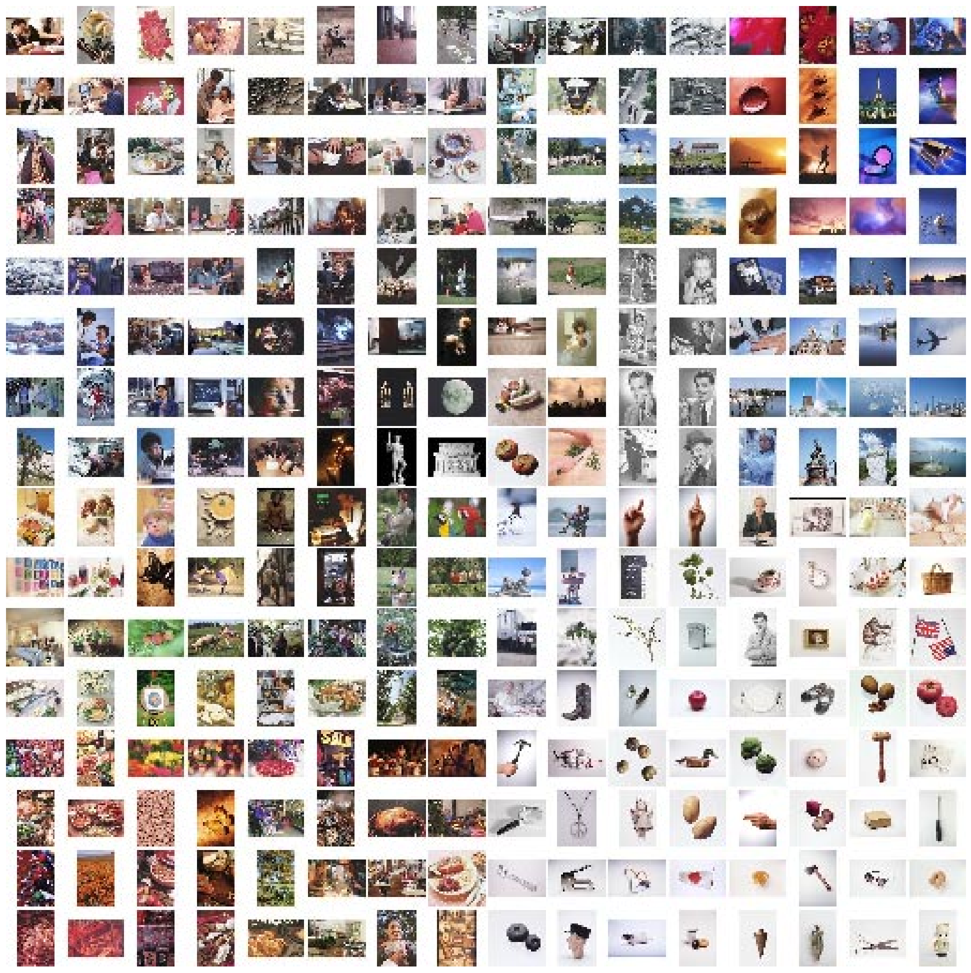


Figure 11. Typical 16x16 layout at level 4 for fast sparse flexible algorithm, with dispersion slight higher than average.

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 figure 11, 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.

6. CONCLUSION

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.

REFERENCES

1. M. 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 and video content: The QBIC system," *Computer* **28**, pp. 23 – 32, September 1995.
2. J. MacCuish, A. McPherson, J. Barros, and P. Kelly, "Interactive layout mechanisms for image database retrieval," in *Proc. of SPIE/IS&T Conf. on Visual Data Exploration and Analysis III*, vol. 2656, pp. 104–115, (San Jose, CA), Jan 31-Feb 2 1996.
3. Y. Rubner, L. Guibas, and C. Tomasi, "The earth mover's distance, multi-dimensional scaling, and color-based image retrieval," in *Proceedings of the ARPA Image Understanding Workshop*, May 1997.
4. M. M. Yeung and B. Liu, "Efficient matching and clustering of video shots," in *Proc. of IEEE Int'l Conf. on Image Proc.*, vol. I, pp. 338–341, (Washington, DC), October 23-26 1995.
5. M. M. Yeung and B.-L. Yeo, "Video visualization for compact presentation and fast browsing of pictorial content," *IEEE Trans. on Circ. and Sys. for Video Technology* **7**, pp. 771–785, October 1997.
6. H. Zhang and D. Zhong, "A scheme for visual feature based image indexing," in *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases III*, vol. 2420, pp. 36–46, (San Jose, CA), February 9-10 1995.
7. J.-Y. Chen, C. A. Bouman, and J. P. Allebach, "Fast image database search using tree-structured vq," in *Proc. of IEEE Int'l Conf. on Image Proc.*, vol. 2, pp. 827–830, (Santa Barbara, CA), October 26-29 1997.
8. J.-Y. Chen, C. A. Bouman, and J. P. Allebach, "Stochastic models for fast multiscale image search," in *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases V*, vol. 3022, pp. 133–144, (San Jose, CA), February 13-14 1997.
9. G. N. Lance and W. Williams, "A general theory of classificatory sorting strategies. i. hierarchical systems," *Computer Journal* **9**, pp. 373–380, 5 1966.
10. P. Sneath and R. Sokal, eds., *Numerical Taxonomy*, Freeman, San Francisco, 1973.
11. A. K. Jain and R. C. Dubes, eds., *Algorithms for Clustering Data*, Prentice Hall, New Jersey, 1988.