IMAGE DATABASE MANAGEMENT USING SIMILARITY PYRAMIDS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jau-Yuen Chen

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 1999

To my parents and my wife, Bao-Lou.

ACKNOWLEDGMENTS

I would like to thank Apple Inc. for their generous financial support. I would also like to thank Mr. John C. Dalton for sharing his brilliant idea and my advisor, Charles A. Bouman for his patient and advice.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES ix

ABSTRACT xiii

1 Introduction 1

2 Multiscale Branch and Bound Image Database Search 3

2.1 Introduction 3

2.2 Multiscale Branch and Bound Search 5

2.2.1 Multiscale distance function 8

2.2.2 Convex distance functions 10

2.3 Feature Vectors 12

2.4 Template Shifting 13

2.5 Experimental Results 15

3 Hierarchical Browsing and Search of Large Image Databases 21

3.1 Introduction 21

3.2 General Approach and Notation 24

3.3 Top-Down and Bottom-up Clustering Algorithms 25

3.3.1 Top-Down Clustering 25

3.3.2 Bottom-up Clustering 26

3.3.3 Fast-Sparse Clustering 29

3.3.4 Binary to quad-tree transformation 32

3.4 Fast Search 34

3.4.1 Best First Search 35

3.4.2 Branch and Bound Search 36

3.5 Database Browsing 37

3.5.1 Quad-Tree to Pyramid Mapping 39
3.5.2 Measures of Pyramid Organization 40
3.6 Experimental Results 42
3.6.1 Search Experiments 42
3.6.2 Browsing Experiments 45
3.7 Conclusion 48
4 Active Browsing using Similarity Pyramids 53
4.1 Introduction 53
4.2 Active Browsing 55
4.3 Pruning 58
4.4 Reorganization 60
4.4.1 Optimized Distance Function 61
4.4.2 Worm Hole Distance 61
4.5 Experimental Results 63
4.6 Conclusion 66
LIST OF REFERENCES 69
APPENDICES 75
Appendix A: Similarity Measures for Images 75
Appendix B: SOM Pyramid Construction 76
Appendix C: Order Statistic 76
Appendix D: Cost function 77
Appendix E: Worm Hole Distance 77
Appendix F: Dispersion for Reorganization 83
VITA 85

LIST OF TABLES

Table Page

2.1 Five similarity metrics used in this research. 10

2.2 Structure of feature vector. 13

3.1 Table of recursion coefficients for standard pairwise agglomerative clustering algorithms. 28

LIST OF FIGURES

Figure Page

2.1 Structure used to compute similarity metric. Both the query and target images are represented by multiscale tree structures where each node contains a feature vector for the corresponding region of the image. The distance function is computed by summing the distance between corresponding nodes of the query and target images. 6

2.2 Branch and bound search example. At the first iteration, all the images in the database are checked at the courses scale. Then the search proceeds to finer scales in order of similarity. 6

2.3 Computational analysis for a set of 30 query images. 16

2.4 Comparison of multiscale and global search using L_1norm with color, texture and edge features. 17

2.5 The effect of adding texture features to multiscale search using L_1norm. 17

2.6 The effect of adding edge features to multiscale search using L_1norm. 17

2.7 The effect of adding texture and edge features to multiscale search using L_1norm. 17

2.8 Each plot compares multiscale and global search using different features and L_1 norm. (a) color features (no texture or edge); (b) 3D color histogram; (c) color cumulative histogram (no texture or edge); (d) color central moments (no texture or edge). 18

2.9 This figure illustrates the effect of multiscale search for some selected examples using a database of $10,000$ images. Notice that the multiscale criteria tends to eliminate matches with different spatial distributions of color, texture and edges. 19

3.1 Algorithm used for flexible clustering with sparse distance matrix. . . 31

3.2 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 equation (3.7). 33

3.3 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. 34

3.4 Algorithm used for both best-first search, and branch and bound search. 35

3.5 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. 38

3.6 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 $l - 1$. 40

3.7 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. 41

3.8 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. 41

3.9 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 $\lambda = 1$. (b) However, the minimax centroid gives uniformly poorer accuracy over a wide range of $0 < \lambda < 1$. . . . 43

3.10 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. 44

3.11 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 8 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. 45

3.12 Speedup versus search accuracy for different number of LBG iteration. 46

3.13 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. 47

3.14 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. 48

3.15 (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 equation 3.12 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}. 49

3.16 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. 50

3.17 Typical 16x16 layout at level 4 for fast sparse flexible algorithm, with dispersion slight higher than average. 51

4.1 An example of a similarity pyramid and its embedded quad-tree. . . 55

4.2 Active Browsing Environment: The top level of the similarity pyramid is shown to the left, and the set of relevant images (relevance set) is shown to the right. The user can incrementally add or remove images from the relevance set as they browse through the database. 56

4.3 The relevance sets/semantic classes of four manually chosen class: Car, Women, Wedding, and Sport. 57

4.4 An illustration of nearest neighbor distance. 58

4.5 An illustration of the "worm hole" distance. The distance from r_x to r_y is ignored. 61

4.6 The recall accuracy for (a) centroid distance, and (b) nearest neighbor distance. The performance of nearest neighbor distance improves with the increasing size of the relevance set. 64

4.7 The theoretic and experimental accuracy versus the rank of the order statistic used for pruning. All experiments use a relevance set of size 16. 65

4.8 The mean of the pruned database size versus the number of images in the relevance set. All examples use a pruning accuracy (recall) of 50%. 65

4.9 The dispersion before and after reorganization: The dispersion from images in the relevance set to images in the same class but not in relevance set. 66

4.10 A result of reorganization with worm hole distance. The images inside rectangle are members of relevance set. 67

F.1 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. 83

ABSTRACT

Chen, Jau-Yuen, Ph.D., Purdue University, May, 1999. Image Database Management using Similarity Pyramids. Major Professor: Charles A. Bouman.

In this work, four major components of image database have been examined: image similarity, search-by-query, browsing environments, and user feedback. We first present a formal framework for designing image similarity and search algorithms. This framework is based on a multiscale representation of both the image data and the associated parameter space. We also define a general form for the distance function which insures that branch and bound search can be used to find the globally optimal match.

We then exploit the techniques of tree structured vector quantization (VQ), branch and bound search, and the triangle inequality to speed-up the search of large image databases. Our method can reduce search computation required to locate images which best match a query image provided by a user. In addition, a free parameter allows computation to be further reduced at the expense of search accuracy.

Our browsing environment is based on a similarity pyramid data structure which hierarchically organizes the database so that it can be efficiently browsed. The similarity pyramid groups similar images together while allowing users to view the database at varying levels of detail. At coarse levels, the similarity pyramid allows users to view the database as large clusters of similar images. Alternatively, users can "zoom into" finer levels to view individual images.

Finally, in order to include users in the "browsing loop", we integrate relevance feedback into the browsing environment. The new browsing environment, which we call active browsing, allows users to dynamically modify the database's organization through the definition of a set of relevant images. The relevance set is then used to adaptively prune and reorganize the pyramid to best suit the user's task.

1. Introduction

The advent of large image databases (>10,000) has created a need for tools which can search and organize images automatically. In the past decade, tremendous attention has been focusing on content-based methods for managing image databases. Most of the initial work in content-based retrieval for image databases has concentrated on the search-by-query approach [1]. Recently, however, people started to look at browsing environments where users can take control of their destiny [2, 3, 4, 5]. In this work, we develop a formal framework for designing multiscale image similarity metrics and search algorithms and then use tree-structured database organizations to both speed-up search-by-query and organize databases for effective browsing.

One method for searching large databases of natural images is to retrieve images with color, texture and edge properties which are similar to a query image provided by the user. Previous approaches to this problem have largely been based on the extraction of a single global feature vector from the query image and each target image in the database. A more natural approach is to directly compare the spatial attributes of the query and target images. However, less attention has been paid to this approach perhaps because it is perceived as being too computationally expensive. Chapter 2 describes a novel framework for designing search algorithms which can identify target images by the spatial distribution of color, edge and texture attributes. This framework is based on a multiscale representation of both the image data, and the associated parameter space that must be searched. We also define a general form for the distance function which insures that branch and bound search can be used to find the globally optimal match. Experimental results indicate that the multiscale approach can improve search performance with minimal computational cost.

While most approaches to image database management have focused on search-by-query [1], the effectiveness of search-by-query can be questionable [3]. 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. In Chapter 3, we 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. This fast-sparse clustering method utilizes a fast search algorithm based on best-first branch and bound search. This search algorithm is designed so that speed and accuracy may be continuously traded-off through the selection of a parameter. 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%.

In Chapter 4, we describe a new approach to managing large image databases which we call active browsing. Active browsing integrates relevance feedback into the browsing environment, so that users can modify the database's organization to suit the desired task. We discuss relevance feedback for the browsing process, and argue that it is fundamentally different from relevance feedback for more traditional search-by-query tasks. We propose two fundamental operations for active browsing: pruning and reorganization. Both of these operation depend on a user defined relevance set which represents the image or set of images desired by the user. We present statistical methods for accurately pruning the database, and we propose a new "worm hole" distance metric for reorganizing the database so that members of the relevance set are grouped together.

2. Multiscale Branch and Bound Image Database Search

This chapter presents a formal framework for designing search algorithms which can identify target images by the spatial distribution of color, edge and texture attributes. The framework is based on a multiscale representation of both the image data, and the associated parameter space that must be searched. We define a general form for the distance function which insures that branch and bound search can be used to find the globally optimal match. Our distance function depends on the choice of a convex measure of feature distance. For this purpose, we propose the L_1 norm and some other alternative choices such as the Kullback-Liebler and divergence distances. Experimental results indicate that the multiscale approach can improve search performance with minimal computational cost.

2.1 Introduction

One method for searching large databases of natural images is to retrieve images with color, texture and edge properties which are similar to a query image provided by the user. Previous approaches to this problem have largely been based on the extraction of a single global feature vector from the query image and each target image in the database. The search is then performed by comparing the query image feature vector to each target image feature vector. This approach is very computationally efficient, but it tends to be limited by the simple global attributes that can be effectively represented in the feature vector.

A more natural approach is to directly compare the spatial attributes of the query and target images. However, less attention has been paid to this approach perhaps because it is perceived as being too computationally expensive. Template matching can be computationally expensive for two reasons. First, spatial comparison of color and texture attributes is intrinsically more computationally expensive then comparison of a single global feature vector. Second, spatial matching assumes that the query and target images are properly aligned. Direct implementation of this alignment step can be very computationally expensive.

To capture information about the spatial distribution of color, Gong *et. al.*[6] divided an image into 9 subareas (3x3) and created a histogram for each of the subareas. Later Zhang *et. al.*[7] concluded that spatial histograms, such as that used by Gong *et. al.*, significantly improving the retrieval rate. Recently, Jacobs *et. al.*[8] proposed a strategy based on direct spatial matching of wavelet coefficients in the query and database images. In each of these cases, the information from all scales was treated as a single feature vector, and the query and target images were match without relative shifting.

This chapter presents a novel framework for image search which allows the use of spatial template matching approaches while retaining the computational efficiency required to search large databases. The key to the approach is a synergistic combination of four techniques: multiscale representation of query and target images; branch and bound search; convex distance metrics based on statistical image models; and multiscale representation of the translation parameters. By using these techniques we are able to efficiently search image databases using a distance metric based on the spatial matching of the color, texture, and edge attributes with 2-D relative translation of the query and target images. This work builds on previous research presented in references [9, 10].

We first show that the branch and bound technique is well suited for searching images in a multiscale framework. This is because the branch and bound algorithm can eliminate poor matches at coarse scales with minimum computation. This insures that each image is only searched to a resolution necessary to determine if it is a potential match candidate. We also note that since the branch and bound algorithm finds the global minimum, this computational savings comes at no cost in the quality of the match.

The key to the effectiveness of the branch and bound algorithm is the formulation of a tight lower bound to the dissimilarity measure. In the multiscale framework, this lower bound can be computed at any resolution, but becomes tighter as the resolution is increased. To formulate this lower bound, we propose a class of dissimilarity criteria formed by summing distances between feature vectors at corresponding locations in the query and target images. We show that by restricting ourselves to distance functions that are convex, we can formulate a useful lower bound on the total dissimilarity measure.

Finally, we discuss the extension of this search method to allow translations of the relative alignment of the query and target images. This is done by quantizing the space of shift parameters in a multiscale pyramid structure. The branch and bound search is then applied to the forest of trees in which the nodes of each tree represents a potential shift of the query image relative to a specific target image.

We present experimental results of searches using a database of $10,000$ images containing a wide variety of natural images including people, wildlife, business, art, landscapes and textures. In these experiments we illustrate how spatial distribution can enhance the quality of the search results without requiring excessive computation. Current versions of the algorithm search the $10,000$ image database in approximately 15 seconds on a Unix workstation.

2.2 Multiscale Branch and Bound Search

Formally, we define the distance function $D(i)$ to be the distance of the i^{th} image in the database to the query image. Our objective is then to search the N images in the database to find the best match.

$$i^* = \arg \min_{1 \leq i \leq N} D(i) \tag{2.1}$$

However, solving (2.1) can be a very difficult problem because with current storage technology N can easily take on values greater than $10,000$. While very simple metrics for $D(i)$ can be rapidly computed and minimized, these metrics may not provide sufficient discrimination to identify the images of interest. Alternatively, more complex metrics may yield better results, but require computation that makes them impractical or impossible to implement for large databases.

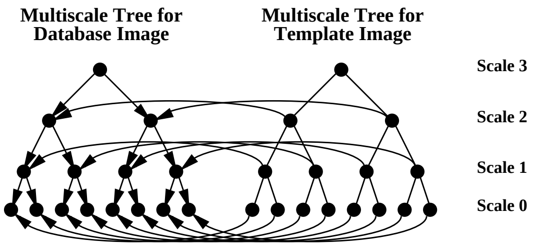

Multiscale Tree for Database Image
Multiscale Tree for Template Image
Scale 3
Scale 2
Scale 1
Scale 0

Fig. 2.1. Structure used to compute similarity metric. Both the query and target images are represented by multiscale tree structures where each node contains a feature vector for the corresponding region of the image. The distance function is computed by summing the distance between corresponding nodes of the query and target images.

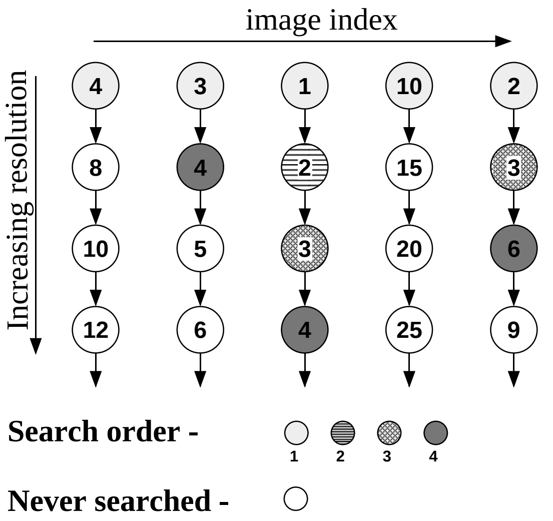


Fig. 2.2. Branch and bound search example. At the first iteration, all the images in the database are checked at the courses scale. Then the search proceeds to finer scales in order of similarity.

Our approach will be to use a multiscale distance function. We will see that such a function can potentially compare fine image differences, but can still be efficiently minimized. The structure of the multiscale distance function is illustrated in Fig. 2.1 for the 1-D case. Both the query and target images are first decomposed into a pyramid of feature vectors. Each level of the tree represents a scale, k ranging from 0 to $L-1$, and a node in the tree at scale k is denoted by $s \in S_k$. At the coarsest scale, the entire image is represented by a single node. As k is reduced, the resolution is increased, and each feature represents a more spatially localized region of the image. The distance function $D(i)$ is computed by adding the distances between corresponding features of the query and target trees.

The difficulty in using a multiscale distance function is that computing the match at fine scales is very computationally expensive. However, this problem can be circumvented by formulating a lower bound to $D(i)$ which is more efficient to compute. More specifically, we will define a function $\underline{D}_k(i)$ such that $D(i) \geq \underline{D}_k(i)$, $D(i) = \underline{D}_0(i)$, and $\underline{D}_k(i)$ can be computed using only nodes at scales greater than or equal to k.

The following algorithm implements branch and bound search. Notice that k_i and $\underline{D}(i)$ are temporary storage variables which contain the current finest resolution of search and lower bound to the distance for image i.

```
1. Set $k_i \leftarrow L - 1$ for all $i$.
2. Compute $\underline{D}(i) \leftarrow \underline{D}_{L-1}(i)$ for all $i$.
3. $i^* = \arg \min_{1 \leq i \leq N} \underline{D}(i)$
4. While $k_{i^*} \neq 0$
    (a) $k_{i^*} \leftarrow k_{i^*} - 1$
    (b) $\underline{D}(i) \leftarrow \underline{D}_{k_{i^*}}(i^*)$
    (c) $i^* = \arg \min_{1 \leq i \leq N} \underline{D}(i)$
```

Figure 2.2 illustrates this branch and bound search with a simple example. At each step, the smallest nodes are searched to the next resolution. If the algorithm terminates with the value i^*, then $D(i^*) = \underline{D}_0(i^*)$ is the smallest distance among all searched nodes. Since each node is a lower bound on nodes that fall below it, $D(i^*)$ must then be the globally minimum solution. When the lower bound is tight, the value of $\underline{D}_k(i)$ may be used to eliminate many poor matches and thereby reduce the size of the search. This search strategy, known as branch and bound or more generally as A^* search[11], leads to an optimal solution to the search problem, but often with dramatically reduced computation. In practice, the minimization of step 4c may be efficiently implemented using a data structure known as a heap. This reduces the time required to find the minimum to order $\log N$. While in theory this heap maintenance dominates the computation, in practice N is generally small enough that evaluation of the distance function dominates computation.

2.2.1 Multiscale distance function

In this section, we define a general form for the distance function which insures that the branch and bound search will be optimal. Figure 2.1 shows that our distance function will be formed by summing the distances between corresponding nodes of the query and target images. Each node for the i^{th} target image is assumed to contain a feature vector, h_{iks}, and a pixel count, N_{ks}, where k is the scale, and s is the index of the node. The set S_k will denote the finite set of all node indices at scale k. Notice that the pixel count, N_{ks}, is assumed to *not* depend on the image i. This is true since the regions represented by each node are generally of the same size and shape. [1] In particular, we will always use $N_{ks} = 4^k$. The tree for the query image is similar, but contains feature vectors, u_{ks}. Notice that u_{ks} is not indexed by i since there is only a single query image. The children of a node $s \in S_k$ are denoted by the set $C(s)$. In the 1-D case, each node has two children, but for images, most nodes have four children.

Given this notation, we will assume that the distance function has the form

$$\begin{aligned} D(i) &= \sum_{k=0}^{L-1} w_k d_k(i) \\ d_k(i) &= \sum_{s \in S_k} N_{k\,s} I(h_{i\,k\,s}, u_{k\,s}) \end{aligned} \tag{2.2}$$

where $w_k \geq 0$ are scalar weights, and $I(\cdot, \cdot)$ is a positive functional which assigned a distance to corresponding nodes in the query and target images.

A basic assumption of our approach is that feature vectors at each node are formed by a weighted average of the feature vectors from child nodes. This assumption is formally stated by the following recursions.

$$\begin{aligned} N_{k+1\,s} &= \sum_{r \in C(s)} N_{k\,r} \\ h_{i\,k+1\,s} &= \sum_{r \in C(s)} \frac{N_{k\,r}}{N_{k+1\,s}} h_{i\,k\,r} \\ u_{k+1\,s} &= \sum_{r \in C(s)} \frac{N_{k\,r}}{N_{k+1\,s}} u_{k\,r} \ . \end{aligned} \tag{2.3}$$

[1]This assumption may cause some difficulties if the shape of the query and target images are different. However, such shape mismatch is generally a problem when comparing spatial attributes of images.

This is a very reasonable assumption for many useful feature vectors. For example, both normalized histograms and moments are formed by spatial averaging, and therefore obey this assumption.

The following property gives a computationally efficient method for computing a lower bound to the distance $D(i)$. This lower bound results from assumptions (2.2), (2.3), and the additional assumption that $I(\cdot,\cdot)$ is convex. The importance of the lower bound, $\underline{D}_k(i)$, is that it is only a function of features at scales k and above. This means that $\underline{D}_k(i)$ may be computed with much less computation than $D(i)$. As discussed in the previous section, a tight lower bound may be used to eliminate many poor matches.

Property 1: Let $D(i)$ be a cost function with the form of (2.2), and let the feature vectors satisfy the recursions of (2.3). Further assume that the functional $I(h,u)$ is a convex function of (h,u). Then the function

$$\underline{D}_k(i) = \sum_{l=0}^{k-1} w_l d_k(i) + \sum_{l=k}^{L-1} w_l d_l(i)$$

is a lower bound on $D(i)$ with $D(i) = \underline{D}_0(i)$.

Proof:

It is enough to show that $d_k(i) \geq d_{k+1}(i)$.

$$\begin{aligned}
d_{k+1}(i) &= \sum_{s\in S_{k+1}} N_{k+1\,s} I(h_{i\,k+1\,s}, u_{k+1\,s}) \\
&= \sum_{s\in S_{k+1}} N_{k+1\,s} I\left(\sum_{r\in C(s)} \frac{N_{k\,r}}{N_{k+1\,s}} h_{i\,k\,s}, \sum_{r\in C(s)} \frac{N_{k\,r}}{N_{k+1\,s}} u_{k\,s}\right) \\
&\leq \sum_{s\in S_{k+1}} N_{k+1\,s} \sum_{r\in C(s)} \frac{N_{k\,r}}{N_{k+1\,s}} I(h_{i\,k\,s}, u_{k\,s}) \\
&= \sum_{s\in S_{k+1}} \sum_{r\in C(s)} N_{k\,r} I(h_{i\,k\,s}, u_{k\,s}) \\
&= \sum_{s\in S_k} N_{k\,r} I(h_{i\,k\,s}, u_{k\,s}) \\
&= d_k(i)
\end{aligned}$$

Distance Type	Formula	Convex	Comments
L_1 norm	$I_{l1}(h,u) = \sum_{i=1}^{M} \lvert h_i - u_i \rvert$	yes	histogram intersection
L_2 norm	$I_{l2}(h,u) = \sum_{i=1}^{M} \lvert h_i - u_i \rvert^2$	yes	
L_p norm	$I_{lp}(h,u) = \sum_{i=1}^{M} \lvert h_i - u_i \rvert^p$	yes	$1 \leq p \leq 2$
Kullback-Liebler	$I_{KL}(h,u) = \sum_{i=1}^{M} h_i \log \frac{h_i}{u_i}$	yes	$\sum_{i=1}^{M} h_i = \sum_{i=1}^{M} u_i = 1$
divergence	$I_d(h,u) = \sum_{i=1}^{M} (h_i - u_i) \log \frac{h_i}{u_i}$	yes	symmetric
generalized divergence	$I_{gdp}(h,u) = \sum_{i=1}^{M} (h_i - u_i)({h_i}^{p-1} - {u_i}^{p-1})$	conjectured for $p > 1$	undefined for $h_i = u_i = 0$

Table 2.1
Five similarity metrics used in this research.

2.2.2 Convex distance functions

In this section, we discuss some potential choices for convex distance functions $I(h,u)$ where h and p are the feature vectors of length M. Table 2.1 summarizes these choices.

In fact, a variety of convex metrics have been proposed for this purpose. Swain and Ballard proposed the use of an L_1 norm, and argued that it had a useful interpretation as a measure of intersection when used with histograms[12]. Alternatively, L_2 norms have also been applied[13, 14], and L_q norms are an obvious generalization. We also note that any of these metrics may be generalized by preprocessing the feature vector with a linear transformation. This leads to the special cases of L_1 norms of the cumulative histogram[14], or weighted L_2 norms[13].

In many applications, the feature vectors are constrained to be positive. For example, normalized histograms (which we will use) must be a member of the set $\Omega^M = \left\{ h \in I\!R^M : h_j \geq 0 \text{ and } \sum_{j=1}^{M} h_j = 1 \right\}$. Under these conditions, there are a number of other interesting convex distance functions to consider. In particular, the Kullback-Liebler distance listed in Table 2.1 is known to be a convex function of (h, u)[15]. Furthermore, it has an interesting statistical interpretation when h and p are the normalized histograms.

Consider a region of the image containing N pixels with a normalized histogram of h. Assume that these pixels are independent and identically distributed with marginal distribution given by the M dimensional vector θ. Under this assumption, the probability of the pixels, x, are given by

$$\log p(x|\theta) = N \sum_{j=1}^{M} h_j \log \theta_j \ .$$

Using this model, we may formulate the distance as the log likelihood ratio between hypothesis 1 that the $\theta = u$ and hypothesis 0 that $\theta \neq u$.

$$\begin{aligned} d &= \log \left(\frac{\max\limits_{\theta \in \Omega^M} p(x|\theta)}{p(x|\theta = u)} \right) \\ &= N \sum_{j=1}^{M} h_j \log \frac{h_j}{u_j} \\ &= N I_{KL}(h, u) \end{aligned}$$

We note that these distance terms are exactly the $d_k(i)$ terms of (2.2).

The Kullback-Liebler distance is useful because differences are effectively normalized to the number of pixels. Therefore, a 10% deviation yields the same difference independently of the absolute value. The divergence distance is a symmetrized version of the Kullback-Liebler distance form by $I_{KL}(h, u) + I_{KL}(u, h)$. In addition, the divergence distance is appropriate to use when $\sum_{i=1}^{M} h_i \neq 1$ or $\sum_{i=1}^{M} u_i \neq 1$. In practice, we have found the divergence distance to perform better.

However, a problem with either the Kullback-Liebler or divergence distance is that they are not well defined when h_i or u_i are 0. One solution is to add a small quantity (we use 10^{-5}) to both h_i and u_i and renormalize them. Perhaps a more elegant solution is the generalized divergence distance proposed in Table 2.1. We conjecture based on numerical evaluation of the Hessian that the generalized divergence is also a convex function of (h, u).

In practice, we have found the L_1 norm to yield accurate matches with minimum computation. However, the divergence appears to be somewhat superior particularly for color histogram features. We believe that the generalized divergence may have the potential for the best overall matching accuracy.

2.3 Feature Vectors

Our feature vectors, h and p, will each consist of three individual feature vectors representing image color, texture and edges respectively. Each of these component feature vectors will be a histogram of the color, texture or edge values in a region of the image. The specific form of these histogram features is described below. Table 2.2 lists the properties of the feature vector discussed below. In practice, many of the histogram entries are empty so the feature vectors can be efficiently stored using run length coding.

The color features are formed by independently histograming the three components of the CIELa^*b^* color space. La^*b^* is a good choice because it is visually uniform and it is also a reasonable approximation to a true opponent color space. In fact, Gargi *et. al.*[16] evaluated six color spaces, and found that La^*b^* performed well. Table 2.2 shows that we chose the number of histogram bins so that the resolution of each bin is approximately $6\Delta E$. In addition, we found that it was important to smooth the histogram so that small color shifts do not excessively effect the match. Therefore, we apply a Gaussian filter kernel with a standard deviation of $6\Delta E$.

The texture feature is 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. These three features are histogramed as shown in Table 2.2.

The edge features are formed by thresholding the edge gradient and then comput-

		Dynamic Range	Number of Bins	Resolution
Color histogram		Dynamic Range	Number of Bins	Resolution
	L^*	$0 \cdots 100$	16	$6\Delta E$
	a^*	$-100 \cdots 100$	32	$6\Delta E$
	b^*	$-100 \cdots 100$	32	$6\Delta E$
Texture histogram		Dynamic Range	Number of Bins	Resolution
	L^*	$0 \cdots 100$	16	$6\Delta E$
	a^*	$0 \cdots 200$	32	$6\Delta E$
	b^*	$0 \cdots 200$	32	$6\Delta E$
Edge histogram		Dynamic Range	Number of Bins	Resolution
	L^*	$0 \cdots 2\pi$	16+1	$\pi/8$
	a^*	$0 \cdots 2\pi$	16+1	$\pi/8$
	b^*	$0 \cdots 2\pi$	16+1	$\pi/8$

Table 2.2
Structure of feature vector.

ing the angle of the edge for all points that exceed the threshold.

$$\Theta_L = \begin{cases} \arctan(D_x L, D_x L) & T_L \geq \sigma_L \\ \emptyset & T_L < \sigma_L \end{cases}$$

The threshold σ_L is given by the standard deviation of the L component for the particular image. The values of Θ_a and Θ_b are computed similarly. These values are then histogramed as shown in Table 2.2. Notice that a 17^{th} bin is reserved for the case of $\Theta = \emptyset$.

2.4 Template Shifting

One limitation of the distance function $D(i)$ is that it can be sensitive to spatial translations of the image regions. Therefore, we extend the search method to allow translations of the query image.

Let Δ be a 2-D displacement vector, and let $u_{ks}(\Delta)$ be the new query tree that results from displacing the query image by Δ. The new distance function is then given by

$$d_k(i, \Delta) = \sum_{s \in S_k} N_{k\,s} I(h_{i\,k\,s}, u_{k\,s}(\Delta))$$

The new search problem requires optimization over both i and Δ which is a computationally intractable problem since Δ is a continuously valued parameter. However, we may discretize the search process by forming a multiscale tree of parameter values $\Delta_{k\,t}$ for $t \in T_k$ where T_k is the discrete set of displacement nodes at scale k. Once again, for $t \in T_k$ with $0 < k < L$, we use the notation $C(t)$ for to denote the children of node t at scale $k-1$. We use the notation $C^n(t)$ to denote the n^{th} descendent of the node t, and we denote the set of all descendents by $C^L(t) = \cup_{l=1}^{k} C^l(t)$ where $t \in T_k$. Using this definition of $d_k(i, \Delta)$, we define a new distance function

$$D(i,t) \quad = \quad \sum_{k=0}^{L-1} w_k d_k(i, \Delta_{0\,t}) \tag{2.4}$$

Assumption: Let $t \in T_k$, and let $r, s \in C^L(t)$ be descendents of t at scales $l, m < k$ respectively. Then we assume that there is a fixed ϵ such that

$$|d_k(i, \Delta_{l\,r}) - d_k(i, \Delta_{m\,s})| \leq \epsilon \ .$$

The value of ϵ may be made small by choosing a sampling period for the parameter space Δ which is sufficiently fine. This is because a sufficiently fine sampling will insure that $||\Delta_{l\,r} - \Delta_{k\,t}||$ is small.

Property 2: Let $D(i,t)$ be a cost function with the form of (2.4), and let the feature vectors satisfy the recursions of (2.3). Further assume that the functional $I(\cdot,\cdot)$ is convex. Define the function

$$\underline{D}_k(i,t) = \sum_{l=0}^{k-1} w_l d_k(i, \Delta_{k\,t}) + \sum_{l=k}^{L-1} w_l d_l(i, \Delta_{k\,t})$$

where $t \in T_k$. Then using Assumption 1,

$$D(i,r) \geq \underline{D}_k(i,t) - \epsilon \sum_{k=0}^{L-1} w_k$$

where $r \in C^k(t)$, and $D(i,r) = \underline{D}_0(i,r)$ for all $r \in T_0$.

Proof:

Let $t \in T_k$ and $r \in C^k(t)$, then

$$\begin{aligned}
D(i,r) - \underline{D}_k(i,t) &= \sum_{l=0}^{k-1} w_l \left(d_l(i,\Delta_{0\,r}) - d_k(i,\Delta_{k\,t})\right) \\
&\quad + \sum_{l=k}^{L-1} w_l \left(d_l(i,\Delta_{0\,r}) - d_l(i,\Delta_{k\,t})\right) \\
&= \sum_{l=0}^{k-1} w_l \left(d_l(i,\Delta_{0\,r}) - d_k(i,\Delta_{0\,r})\right) \\
&\quad + \sum_{l=0}^{k-1} w_l \left(d_k(i,\Delta_{0\,r}) - d_k(i,\Delta_{k\,t})\right) \\
&\quad + \sum_{l=k}^{L-1} w_l \left(d_l(i,\Delta_{0\,r}) - d_l(i,\Delta_{k\,t})\right) \\
&\geq \sum_{l=0}^{k-1} w_l\,(0) + \sum_{l=0}^{k-1} w_l\,(-\epsilon) + \sum_{l=k}^{L-1} w_l\,(-\epsilon) \\
&= -\epsilon \sum_{l=0}^{L-1} w_l
\end{aligned}$$

2.5 Experimental Results

In the following experimental results, we our algorithm to search a image data base containing $10,000$ images. We applied a weighting of $w_k = 1/2^k$ at each scale $k \geq L-3$. We only searched to a 4×4 resolution which is equivalent to setting $w_k = 0$ for of $k < L-3$.

Figure 2.3 shows the computational advantages of the multiscale search algorithm. The first graph shows that only a small number of images are search to finer resolutions. As the resolution increases, the number of possible shifts of the query image rapidly increases, so the number of positions searched does not decrease as rapidly as the number of images searched. The third graph shows the number of nodes compared. This measure is approximately proportional to computation; so we see that the computational cost is approximately constant at each resolution. The final graph shows a histogram of computation time for search of the $10,000$ images on an HP755/100. Average search time is approximately 15 seconds.

The large database is useful for making realistic assessments of computational

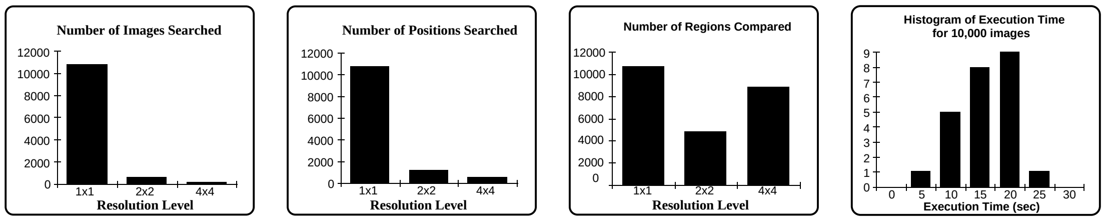


Fig. 2.3. Computational analysis for a set of 30 query images.

speed and qualitative performance. However, it is difficult to make accurate measurements for match quality. To make these more accurate measurements, we used the methodology suggested by Frese *et. al.*[17] Frese *et. al.* collected data from subjects on smaller 200 image randomized subsets of the full database. Our algorithm was then used to rank the complete set of 200 images. Given this ranking, one can compute the probability of the subjects "best" match falling among the first n images as ranked by the algorithm. This plot of probability of detection versus $\log n$ gives a useful measure of algorithm performance.

Figure 2.4 shows the increased search accuracy of the multiscale search versus the search using only the single coarse scale feature vector. It shows a 5 percent improvement for images ranking from 3 to 10 out of 200. Figures 2.5 2.6 and 2.7 compare the search results using various combinations of color, texture and edge features. In particular, Fig. 2.7 indicates that texture and edge features are important when used with color features.

Figure 2.8 illustrates how multiscale search improves match quality using a variety of alternative feature vectors.

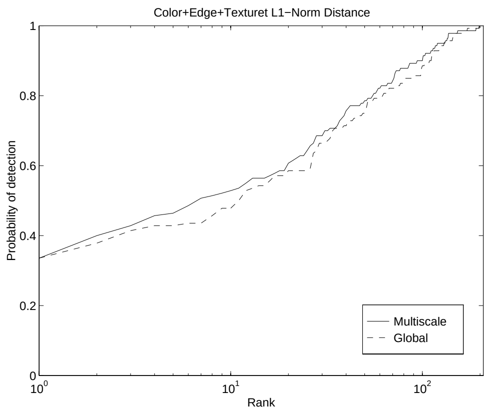


Fig. 2.4. Comparison of multiscale and global search using L_1norm with color, texture and edge features.

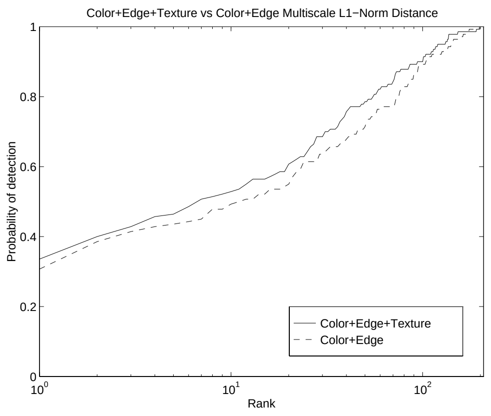


Fig. 2.5. The effect of adding texture features to multiscale search using L_1norm.

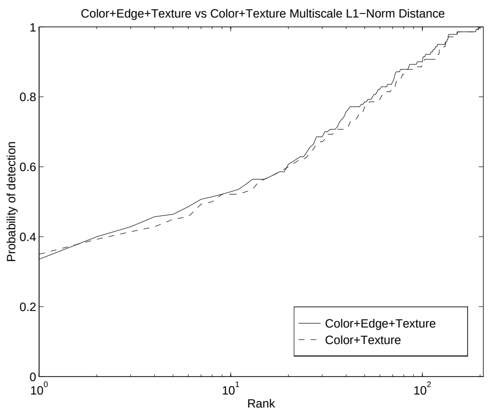


Fig. 2.6. The effect of adding edge features to multiscale search using L_1norm.

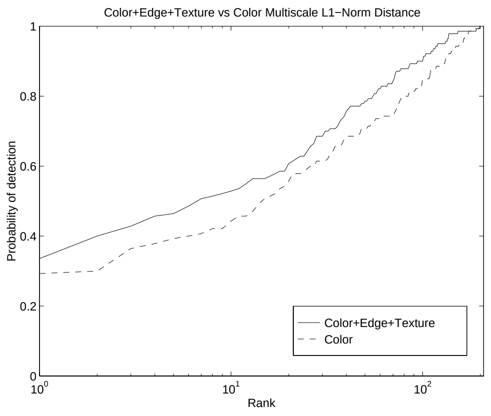
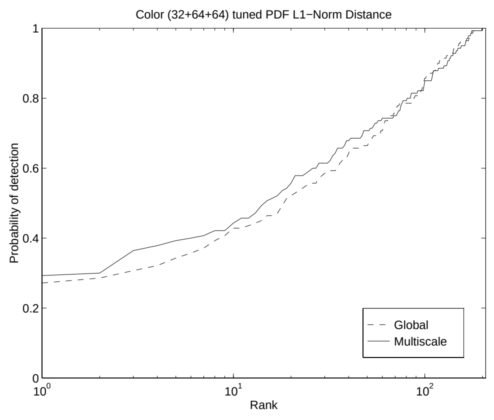
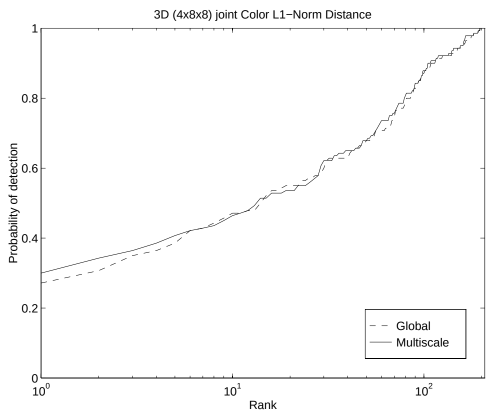


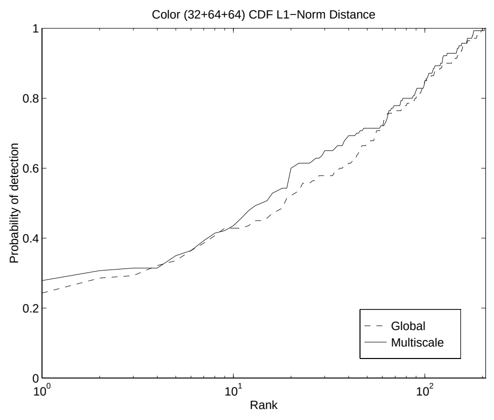
Fig. 2.7. The effect of adding texture and edge features to multiscale search using L_1norm.



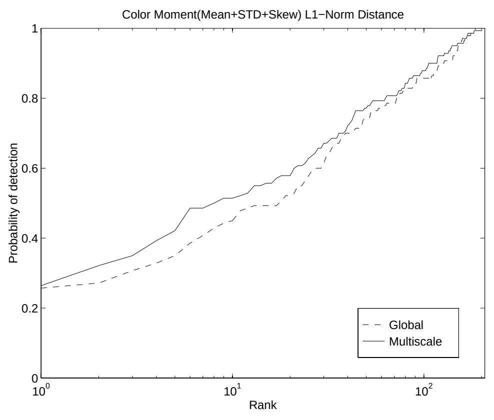
(a)



(b)



(c)



(d)

Fig. 2.8. Each plot compares multiscale and global search using different features and L_1 norm. (a) color features (no texture or edge); (b) 3D color histogram; (c) color cumulative histogram (no texture or edge); (d) color central moments (no texture or edge).

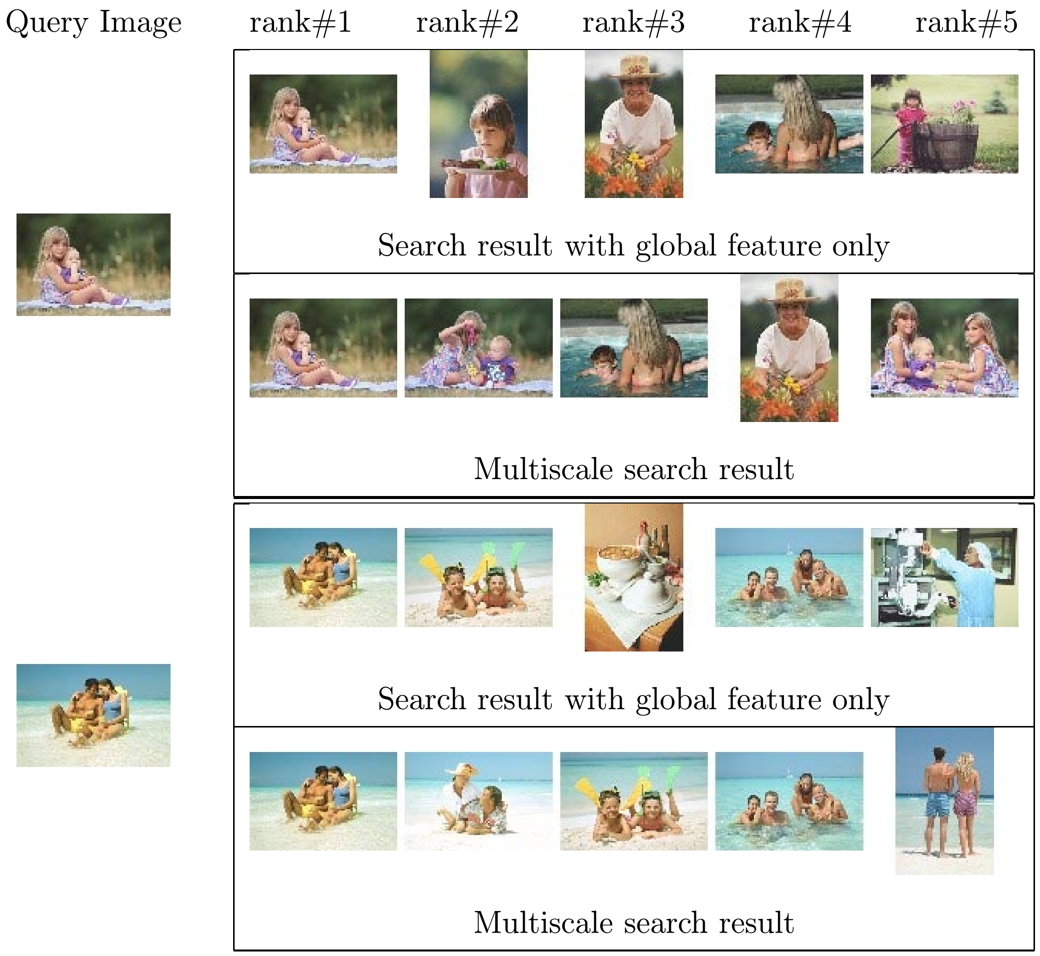


Fig. 2.9. This figure illustrates the effect of multiscale search for some selected examples using a database of 10, 000 images. Notice that the multiscale criteria tends to eliminate matches with different spatial distributions of color, texture and edges.

3. Hierarchical Browsing and Search of Large Image Databases

3.1 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 chapter 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 [3]. 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* [3] used MDS to organize images returned by queries while Rubner, Guibas, and Tomasi [4] 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, Yeo and Liu studied the application of clustering methods to the organization of video key frames [18, 19]. 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, Squire and Pun have applied heirarchical clustering to organize an image database into visually similar groupings [20].

Zhang and Zhong [2] 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 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 i^{th} 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) [21], and key word search [22, 23]. More recently, such techniques have been applied to image database search [24, 25].

Perhaps the most widely studied method for speeding minimum distance search is the k-d tree [26, 27, 28]. 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 $K >> \log_2 N$, 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 [29]. This chapter 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, Kelley and Vincent [30], White and Jain [31], and Kurniawati, Jin and Shepherd [32] have considered similar approaches for image database search.

In this chapter, 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% [33].
- A hierarchical browsing environment which we call a similarity pyramid that efficiently organizes databases so that similar images are located nearby each other [34].

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 [29, 30, 31, 32]. The best-first approach is optimal in the

[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.

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/accuracy 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 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.

3.2 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 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 Appendix A. More generally, the feature vectors could represent other data types, such as audio, but we will only consider images in this chapter. 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 $C_i = \{i\}$.

3.3 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 3.3.3 we will introduce a fast-sparse clustering algorithm which implements the standard flexible agglomerative (bottom-up) clustering algorithm [35], 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 3.5.

3.3.1 Top-Down Clustering

Top-down methods work by successively splitting nodes of the tree working from the root to the leaves of the tree [36]. We will use the K-means [37] or equivalently

LBG [38] 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 $K$, set $C_r \leftarrow \{x \in C : d(x, q_r) < d(x, q_j)$ for
        $j \neq r$ $\}$.
    (b) For $r = 1$ to $K$, set $q_r \leftarrow \text{centroid}(C_r)$.
   }
```

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.

$$\text{Mean:} \quad q_r = \frac{1}{|C_r|} \sum_{s \in C_r} x_s \tag{3.1}$$

$$\text{Median:} \quad [q_r]_i = \text{median}_{s \in C_r} \left([x_s]_i \right) \tag{3.2}$$

$$\text{Minimax:} \quad q_r = \arg\min_q \left\{ \max_{s \in C_r} \ d(q, x_s) \right\} \tag{3.3}$$

Here $[q_r]_i$ refers to the i^{th} 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, $|C_r| \leq \lceil \frac{N}{K} \rceil$, to step 3.(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.

3.3.2 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 [39].

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 [36, 40]. 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 clus-

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 $\beta < 0$

Table 3.1
Table of recursion coefficients for standard pairwise agglomerative clustering algorithms.

tering methods [35].

$$d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} - \gamma |d_{hi} - d_{hj}| \tag{3.4}$$

Here α_i, α_j, β and γ are coefficients which depend on some property of the clusters. Table 3.1 lists the particular choices of these coefficients for a number of standard clustering algorithms [36, 40].

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 [35] which uses the update rule

$$d_{hk} = \frac{1-\beta}{2} d_{hi} + \frac{1-\beta}{2} d_{hj} + \beta d_{ij} \tag{3.5}$$

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,

$$d_{hk} = d_{hi} + d_{hj} - d_{ij} \ . \tag{3.6}$$

3.3.3 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 3.4.

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 (3.6) 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 (3.6) 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

$$d_{hk} = \begin{cases} d_{hi} + d_{hj} - d_{ij} & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are available} \\ \hat{d}_{hi} + d_{hj} - d_{ij} & \text{if } d_{hi} \text{ is missing} \\ d_{hi} + \hat{d}_{hj} - d_{ij} & \text{if } d_{hj} \text{ is missing} \\ \text{Do not compute} & \text{if both } d_{hi} \text{ and } d_{hj} \text{ are missing} \end{cases} . \tag{3.7}$$

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

$$\tilde{d}_{hk} = \tilde{d}_{hi} + \tilde{d}_{hj} \tag{3.8}$$

in place of the recursion of (3.6). 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.

$$\tilde{d}_{hk} = \sum_{m \in C_h} \sum_{n \in C_k} d(x_m, x_n) \tag{3.9}$$

We may use (3.9) to approximate the missing term $\hat{d}_{hi}$ of (3.7). 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

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

```
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 (3.7) with $\hat{d}_{hi} = \max(n_h r_h \, , \, n_i r_i)$
  }
```

Fig. 3.1. Algorithm used for flexible clustering with sparse distance matrix.

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

$$\hat{d}_{hi} \stackrel{\triangle}{=} \max\left(n_i \sum_{m \in C_h} r_m \, , \, n_h \sum_{n \in C_i} r_n \right) \tag{3.10}$$

were the summation terms may be recursively computed as part of the clustering algorithm. Figure 3.1 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 3.4 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 3.4 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 3.3.1.
2. Construct the sparse distance matrix using the fast search algorithm of Section 3.4 together with the top-down tree.
3. Apply the sparse clustering algorithm of Figure 3.1.

Figure 3.2 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 i^{th} column and then across the i^{th} row. We will refer to this list of entries as the i^{th} 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 (3.7). For case A, both elements are present. The result of (3.7) is stored in $d_{2\,i}$, and the entry $d_{1\,j}$ is deleted. For case B, the element $d_{1\,j}$ is missing; so it must be computed using $\hat{d}_{2\,j}$. The result of (3.7) is then stored in $d_{2\,i}$. For case C, a new entry must be inserted into location $d_{3\,i}$, and the entry at $d_{3\,j}$ must be removed. For case D, nothing is done.

3.3.4 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.

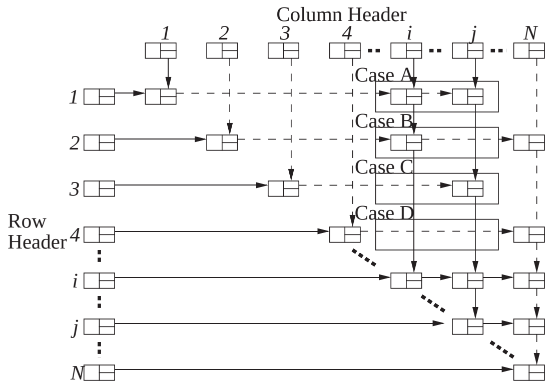


Fig. 3.2. Data structure used for storing sparse matrix for clustering. Both rows and columns have the structure of linked lists. Dark lines are used for links along columns, while doted lines link rows. Each case represents one of the four update possibilities for equation (3.7).

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 quad-tree case (K=4) since this will be important for the browsing application of Section 3.5.

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

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

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 3.3 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 $P \in \mathbf{P}(s, K)$.

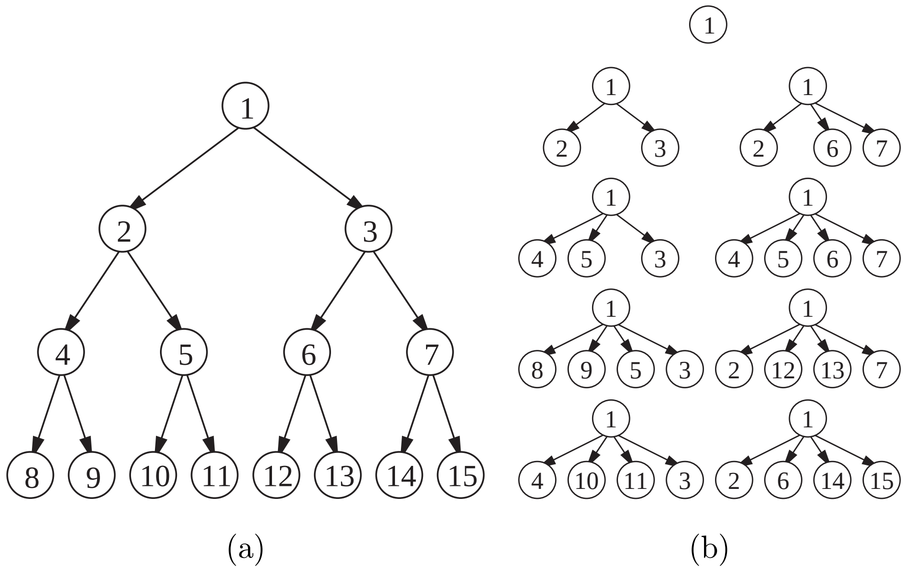


Fig. 3.3. All 4 partitions of a binary tree node $s = 1$. (a) Original binary tree with root node $s = 1$. (b) The 9 unique partitions of the node $s = 1$. Each partition contains 4 or less nodes that partition the original node.

In order to minimize tree depth, we use the maximum number of elements in the children nodes as the cost function. That is

$$P^* = \min_{P \in \mathbf{P}(s,K)} \left\{ \max_{j \in P} \ n_j \right\} \tag{3.11}$$

where $n_j = |C_j|$.

3.4 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 3.3.3 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 structure. Our method couples a best-first

```
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. 3.4. Algorithm used for both best-first search, and branch and bound search.

implementation of branch and bound search with the structure imposed by a top-down tree of Section 3.3.1 to substantially reduce computation.

3.4.1 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.

Figure 3.4 show the algorithm used for best first search of trees [11]. 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. 3.4 may be efficiently implemented using a data structure known as a heap [41] 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.

3.4.2 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 [11] and the solution is guaranteed to be the global minimum. Then $\underline{d}_s$ must have the property that

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

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 [29]. 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 $i \in C_s$

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

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 $0 \leq \lambda \leq 1$ 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 [42].

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 10 most similar images, these 10 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.

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

Figure 3.5 illustrates a simple example of a similarity pyramid and its associated quad-tree. Figure 3.5(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 3.5(b) shows level $l = 1$ of the similarity pyramid while Fig. 3.5(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 it allows a user to keep relative orientation when moving between levels of the pyramid.

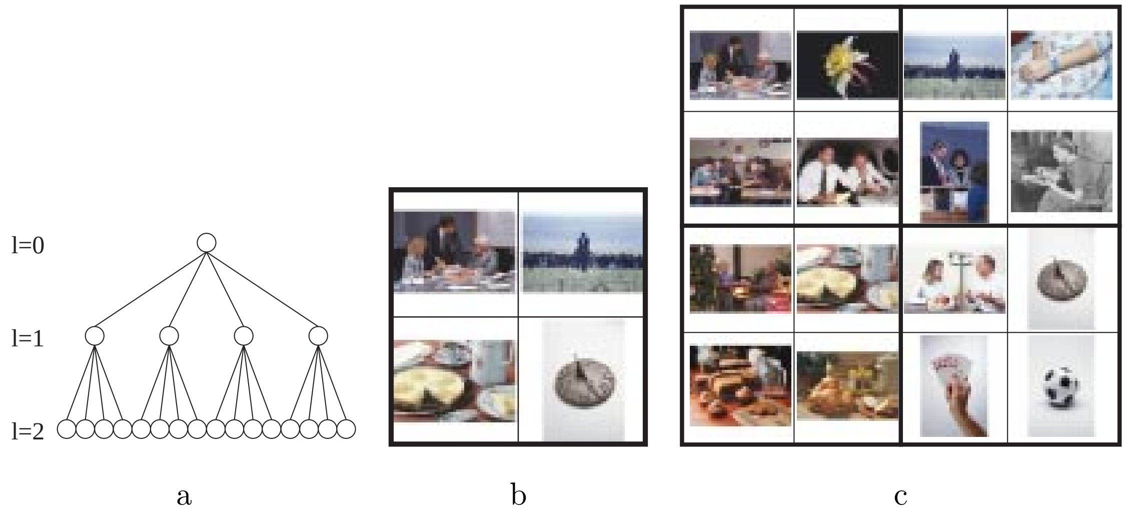


Fig. 3.5. This figure illustrates how nodes are mapped from the quad-tree to the similarity pyramid. There are 24 possible mappings of 4 children nodes to four pyramid nodes. The mapping is chosen to maximize spatial smoothness in image characteristics.

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 (3.1).

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 3.13 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 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.

We will primarily use the fast-sparse clustering algorithm of Section 3.3.3 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 quad-tree. 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.

3.5.1 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 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

$$\text{Total Cost} = E_{inter} + E_{intra} + E_{extern} \tag{3.12}$$

where the three terms represent inter-cluster, intra-cluster, and external costs in node placement. Figure 3.6 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 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

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

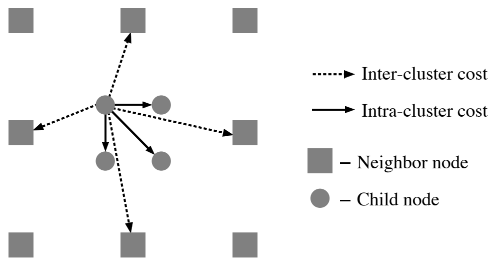


Fig. 3.6. This figure illustrates the inter-cluster and intra-cluster cost terms used to organize images in the pyramid. The neighbor nodes are at level l of the pyramid, while the children nodes are at level $l-1$.

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 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.

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

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.

3.5.2 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

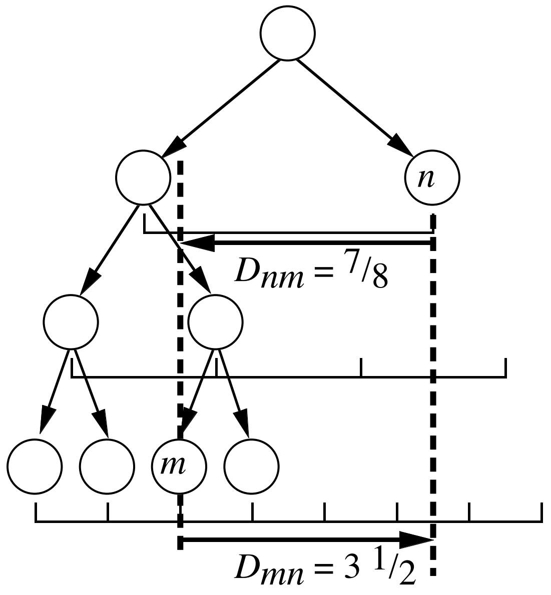


Fig. 3.7. This figure illustrates the distance measured between two images in the similarity pyramid. The distance is an asymmetric function because it is measured relative to the first image.

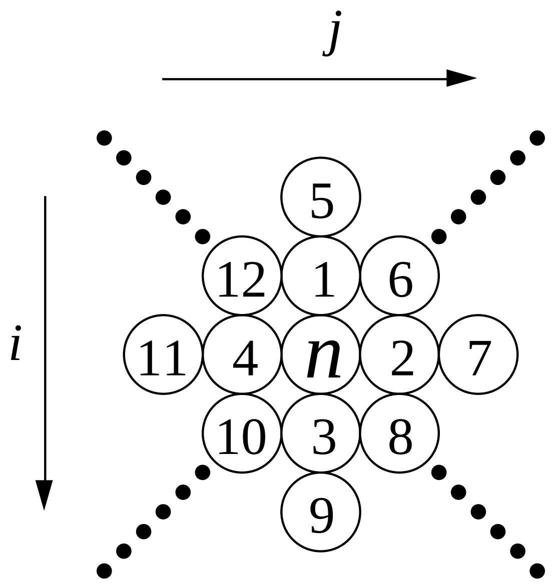


Fig. 3.8. An example of a dense packing of the M closest matches to image n. The M images are arranged in a diamond shape such that the physical distance $D_{n\pi(n,m)}$ is nondecreasing with respect to m.

similar images in the pyramid. Figure 3.7 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

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

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

$$\frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)} \tag{3.13}$$

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. 3.8. 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

$$\underline{D}(M) = \frac{1}{M} \sum_{m=1}^{M} L(m) \tag{3.14}$$

Combining the expression of (3.13) and the lower bound of (3.14) results in a normalized measure of dispersion.

$$Dispersion = \frac{1}{NM\underline{D}(M)} \sum_{n=1}^{N} \sum_{m=1}^{M} D_{n\pi(n,m)} \tag{3.15}$$

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

3.6 Experimental Results

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. Appendix A describes the 211 element feature vector that was extracted for each image [43]. In all cases, the L_1 norm was used as the distance metric.

Section 3.6.1 presents results of search experiments and experimentally determines the tradeoffs between search accuracy and speed. Section 3.6.2 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 3.4 to the problem of pyramid design, and measure the computational savings of this approach.

3.6.1 Search Experiments

In this section, all trees were designed using a coarse-to-fine procedure as described in Section 3.3.1. Unless otherwise stated, a fan-out of 10 was used ($K = 10$), centroids were computed using feature means as in (3.1), 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.

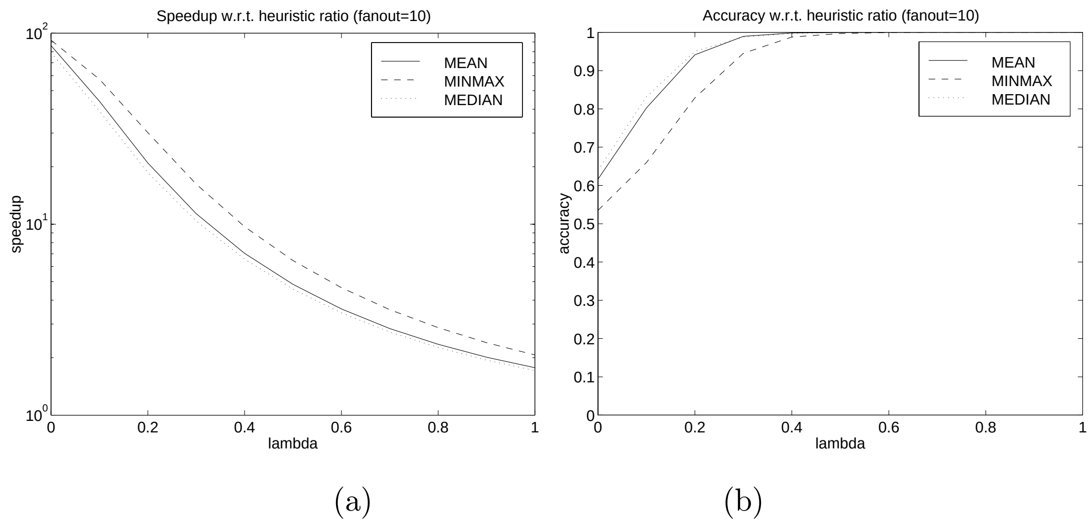


Fig. 3.9. Speed-up and accuracy of search versus λ. Each plot is for a different cluster centroid. (a) Notice that the minimax centroid gives the best speed-up for exact search at $\lambda = 1$. (b) However, the minimax centroid gives uniformly poorer accuracy over a wide range of $0 < \lambda < 1$.

Unless otherwise stated, we selected the 10 best image matches after searching for 20 images. (i.e. We searched for 10 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. 3.4. 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** - Figures 3.9(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 $\lambda = 1$ (exact search) the minimax centroid gives better speed-up than the mean and median [32]. However, the speed-up for exact search is limited to approximately 2 which is much less than can be achieved with approximate search.

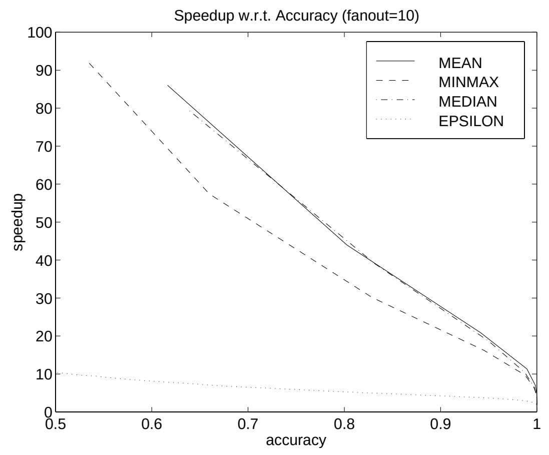


Fig. 3.10. Speedup versus search accuracy. This plot shows that over a wide range of search accuracy the mean and median centroids are preferable to the minimax.

However, Fig. 3.10 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. Figure 3.10 also compares the fast search algorithm to a standard approximate search method known as $(1+\epsilon)$ search [42]. 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 - Figure 3.11(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(fan-out)(tree depth). However, Fig. 3.11(a) indicates that a fan-out less than 8 can substantially reduce performance.

Figure 3.11(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.

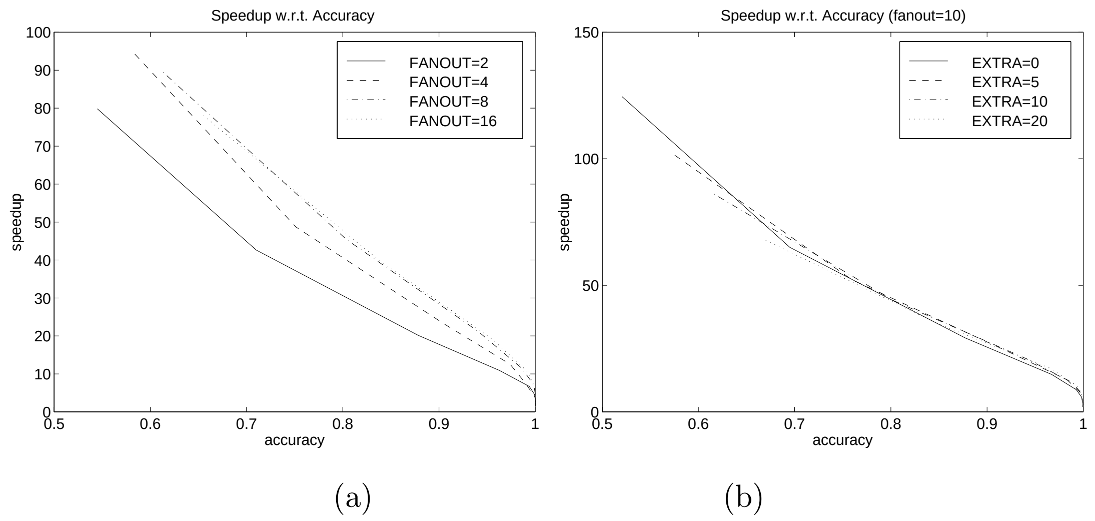


Fig. 3.11. Speedup versus search accuracy for different tree fan-outs and different numbers of extra returned images. (a) Higher tree fan-outs yields better performance, but requires more computation for tree construction. A fan-out of 8 appears sufficient. (b) Speed-up versus search accuracy for different numbers of extra returned images. Returning extra images can improve the speed-up/accuracy tradeoff, but 50% extra returns appears sufficient.

Figure 3.11(b) indicates that returning 5 extra images is sufficient to gain most of the benefit.

LBG iteration - Figures 3.12 shows that LBG iteration can improve performance. However, the effect is rather weak after 4 iteration. Furthermore, the larger the number of return images, the smaller the improvement of multiple LBG iterations.

3.6.2 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.

Figure 3.13 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

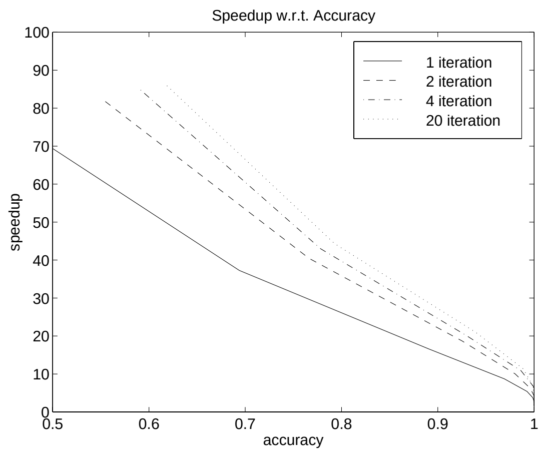


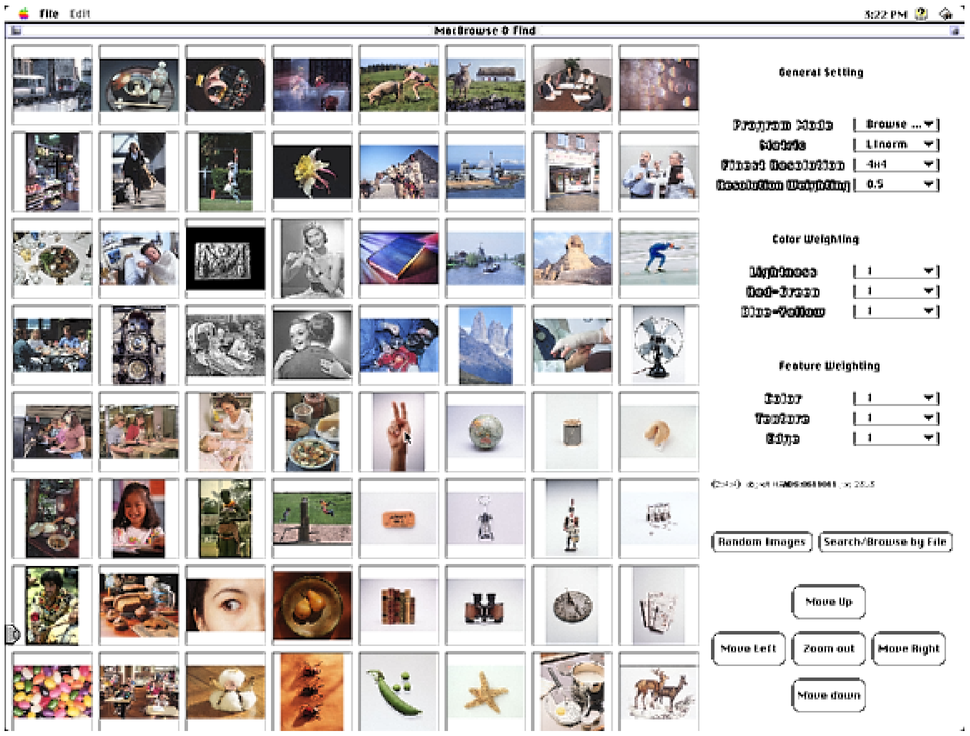
Fig. 3.12. Speedup versus search accuracy for different number of LBG iteration.

database, or move up the pyramid. The user can move down the pyramid by "double clicking" on a specific image icon.

In Figure 3.14, 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.

Figure 3.15(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.

Figure 3.15(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}.



General Setting

Program Mode	Browse ...
Metric	L1norm
Finest Resolution	4x4
Resolution Weighting	0.5

Color Weighting

Lightness	1
Red-Green	1
Blue-Yellow	1

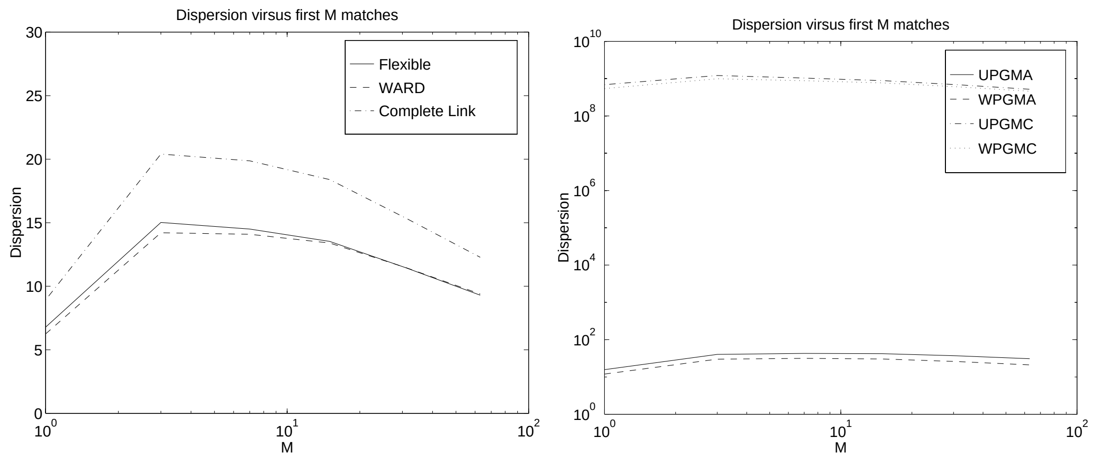
Feature Weighting

Color	1
Texture	1
Edge	1

[illegible]

Fig. 3.13. 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.

In Figure 3.16, 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.



(a) (b)

Fig. 3.14. 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.

In Figure 3.17, 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.

3.7 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 2.

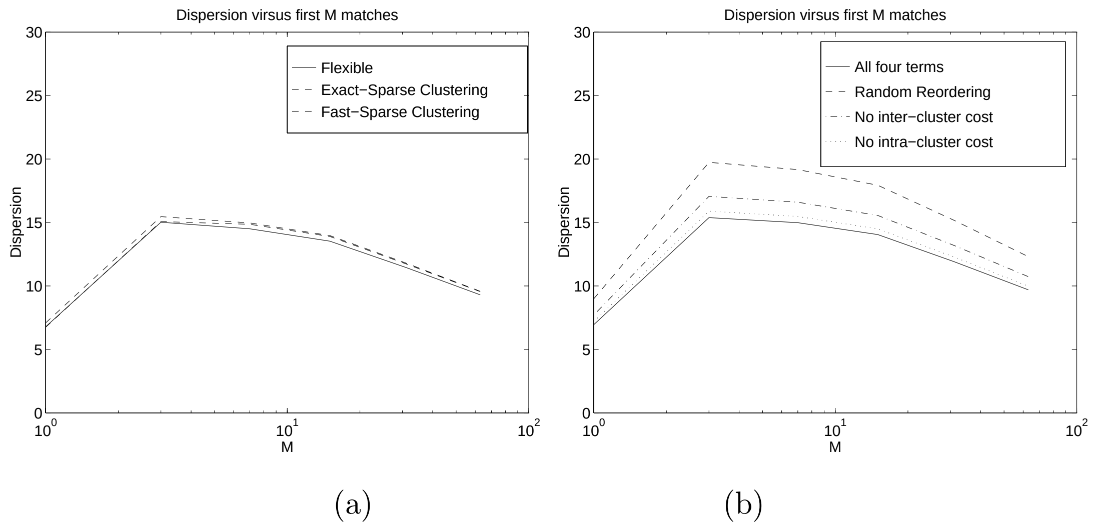


Fig. 3.15. (a) Comparison of flexible clustering with full and sparse matrices and fast-sparse method. All three variations of flexible clustering seems to perform similarly. Effect of reordering algorithm. (b) The performance of reordering algorithm using equation 3.12 is much better than using random reordering. It also shows that the effect of inter-cluster cost, E_{inter} and intra-cluster cost, E_{intra}.

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.

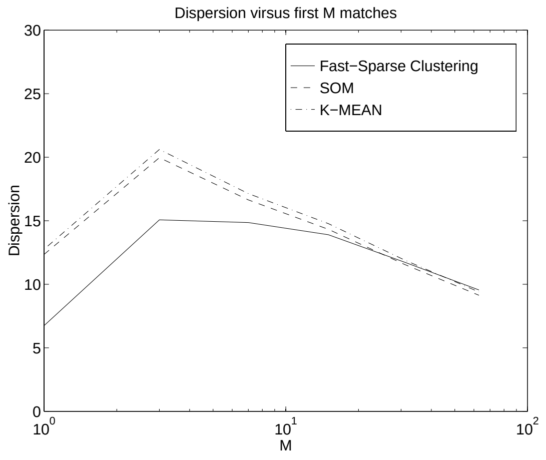


Fig. 3.16. Dispersion versus M for K-means and SOM trees and fast-sparse flexible. Apparently, SOM outperforms K-means everywhere. The flexible method has better performance for low values of M which seem to be most important in practice.

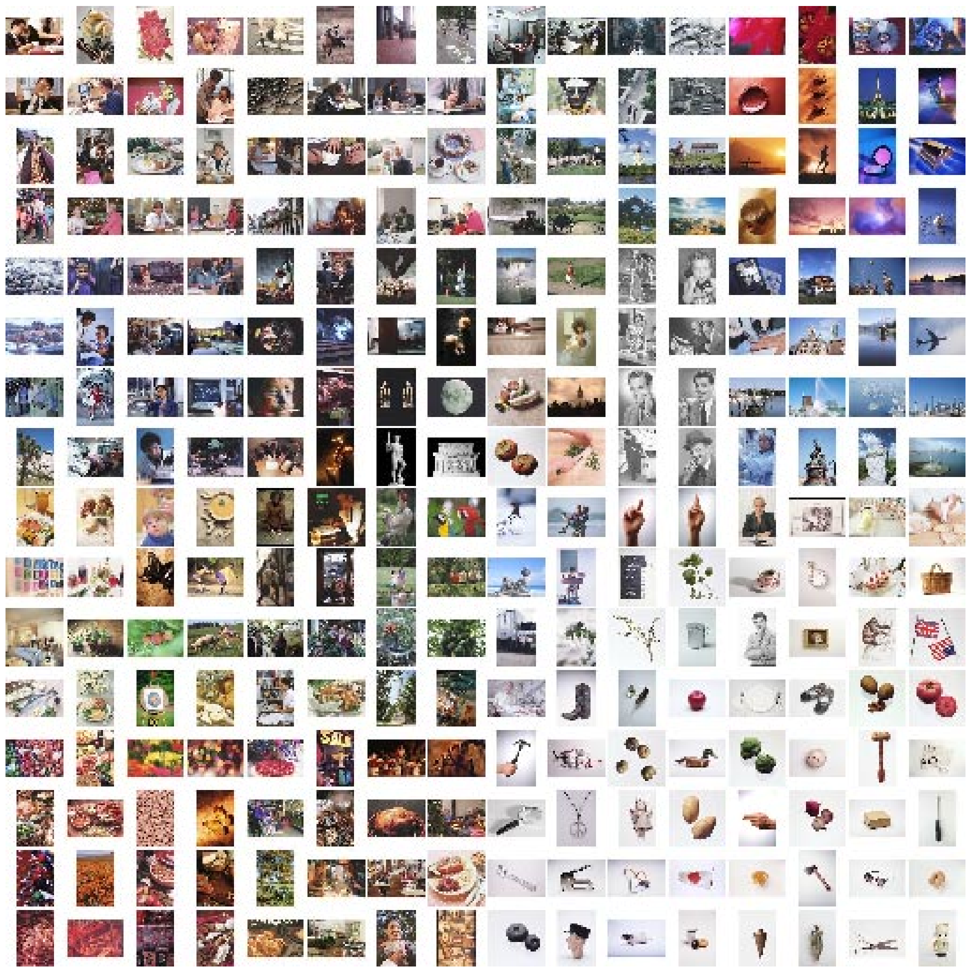


Fig. 3.17. Typical 16x16 layout at level 4 for fast sparse flexible algorithm, with dispersion slight higher than average.

4. Active Browsing using Similarity Pyramids

4.1 Introduction

Many approaches for content based management of image databases have focused on query-by-example methods [1] in which an example image is presented and the database is searched for images with similar visual content. However, query-by-example techniques tend to quickly converge to a small set of images that may not be of interest to the user [3].

Browsing environments offer an alternative to conventional query-by-example, but have received much less attention. Zhang and Zhong [2] proposed a hierarchical self-organizing map (HSOM) which used the SOM algorithm to organize a complete database of images into a 2-D grid. MacCuish *et al* [3] used multidimensional scaling (MDS) to organize images returned by queries while Rubner, Guibas, and Tomasi [4] used MDS for direct organization of a complete database. Most recently, Craver, Yeo and Yeung [5] described a browsing method based on space-filling curves. A disadvantage of both SOM and MDS is that they are vary computationally intensive when applied to large databases of images. In addition, the MDS algorithms and space-filling curves approach do not impose any hierarchical structure on the database. Hierarchical database structure is important because it can allow the user to move quickly from one region of the database to another.

Previously, we proposed a hierarchical data structure for browsing large databases of images which we call a similarity pyramid [34, 44]. The similarity pyramid organizes large image databases into a three dimensional pyramid structure. Each level of the similarity pyramid contains clusters of similar images organized on a 2-D grid. As users move down the pyramid, the clusters become smaller, with the bottom level of the pyramid containing individual images. In addition, users can pan across at a single level of the pyramid to see images or image clusters that are similar. In previous research[34, 44, 33], we described efficient algorithms for constructing good similarity pyramids. The work used a fixed image dissimilarity measure, and therefore did not treat the issues of user feedback in the browsing process.

While relevance feedback has long been recognized as useful for data retrieval problems [45], more recently it has attracted attention in content based image retrieval. Minka and Picard's FourEyes were among the first to apply relevance feedback to the problem of image search [46, 47]. Cox, Miller, Omohundro, and Yianilos's PicHunter [48] used the relevance feedback to continuously update the probability that each image in the database is the desired target. Both Taycher, La Cascia, and Sclaroff's ImageRover [49], Rui, Huang, and Mehrotra's MARS [50], and Santini and Jain [51] used relevance feedback algorithms to optimize the similarity measure used for searching the database. Wood, Campbell, and Thomas's IDQS [52] identified centroids in the feature space by clustering the relevance feedback. This approach is interesting because it explicitly accounts for the fact the desired images may not fall into a single region of the feature space. In MetaSEEk user feedback is used to select among different search engines rather then optimizing the performance of a single search method [53].

Importantly, none of the previous studies have considered how relevance feedback can be incorporated in the browsing process. Instead, the focus has been on the problems of search-by-query.

In this chapter, we describe an *active browsing* approach which incorporates relevance feedback into the browsing environment so that users can more effectively manage and search large databases of images or video [54, 55, 56, 57]. We argue that relevance feedback for browsing is fundamentally different from relevance feedback technique used for search-by-query tasks. Our active browsing environment uses a similarity pyramid to organize the database into a 3 dimensional structure that the user can move through. But in addition, the environment allows the user to give feedback through the selection of a set of desirable images which we call the relevance

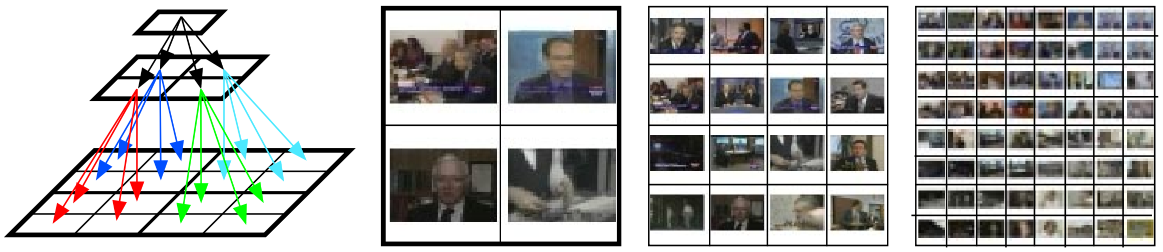


Fig. 4.1. An example of a similarity pyramid and its embedded quad-tree.

set. The relevance set is then used to adaptively prune and reorganize the pyramid to best suit the user's task. In order to perform these basic pruning and reorganization functions, we use statistical estimation techniques based on cross-validation. The cross-validation approach is shown to give reliable performance independently of the database content and specific choice of similarity measures. We also introduce a dissimilarity measure which groups images from the relevance set together in the similarity pyramid. We call this new dissimilarity measure the "worm hole" distance because it connects all points in relevance set. We prove that the worm hole distance is a metric, and we show that use of the worm hole distance can substantially improve the organization of the browsing environment.

4.2 Active Browsing

The structure of a similarity pyramid is illustrated in Fig. 4.1. Each level of the similarity pyramid contains clusters of similar images organized on a 2-D grid. As one moves down the pyramid, the clusters become smaller, with the bottom level of the pyramid containing individual images. In addition, users can pan across at a single level of the pyramid to see images or image clusters that are similar.

The similarity pyramid combines the advantages of both searching and browsing. In fact, at the bottom levels of the pyramid, an image is surrounded by the images that would be returned by a simple query-by-example. However, the pyramid structure allows the user to "zoom out" so that they can see a larger variety of query returns, or pan across to move toward a different set of images. For a details on the design of similarity pyramids, see [34, 44].

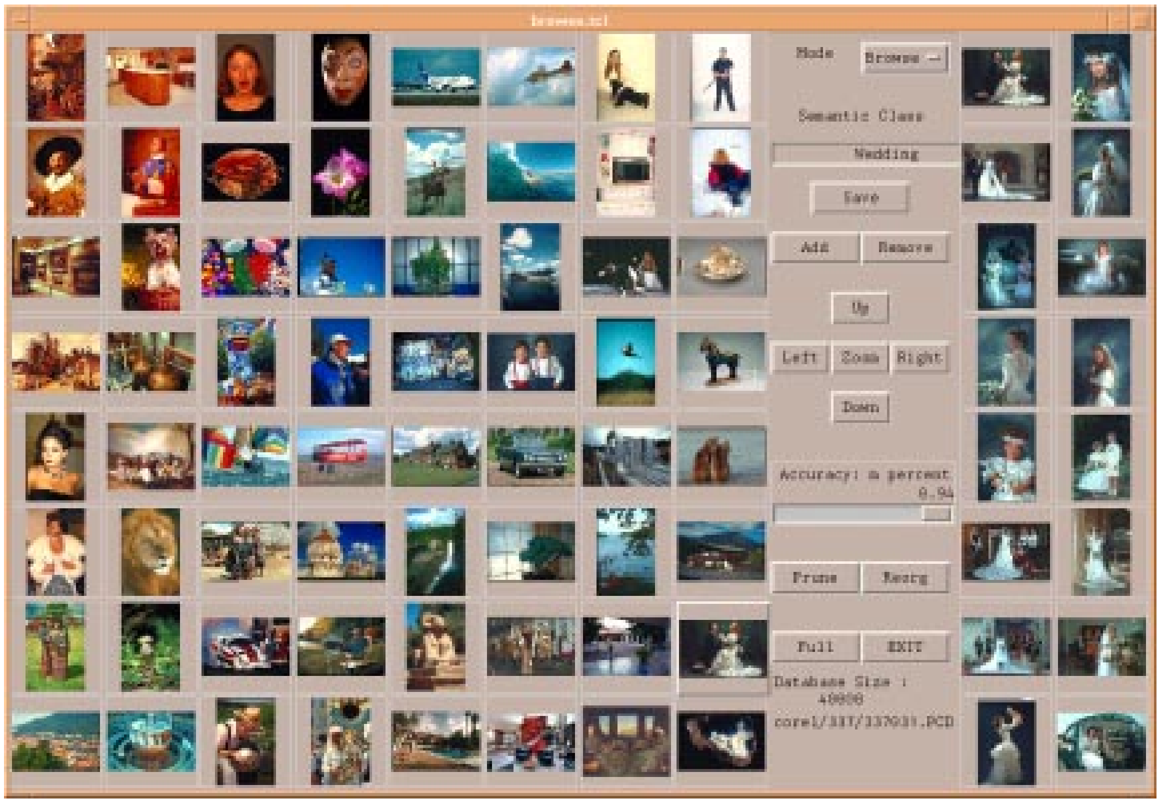


Fig. 4.2. Active Browsing Environment: The top level of the similarity pyramid is shown to the left, and the set of relevant images (relevance set) is shown to the right. The user can incrementally add or remove images from the relevance set as they browse through the database.

In this work, we combine relevance feedback with the browsing environment of a similarity pyramid. We will incorporate relevance feedback into two basic browsing functions: pruning and reorganization. Pruning removes images from the database that are not likely to be of interest to the user, and reorganization changes the structure of the similarity pyramid to facilitate the user's search.

Figure 4.2 shows the active browsing environment presented to the user. On the left is a region of similarity pyramid at a particular level of the pyramid. The user can pan across (up, down, left, or right) at a fixed level of the pyramid. The user can also "zoom" out to a courser level of the pyramid or zoom in by simply clicking on a desired image. On the right of Figure 4.2 is the set of relevant images which we

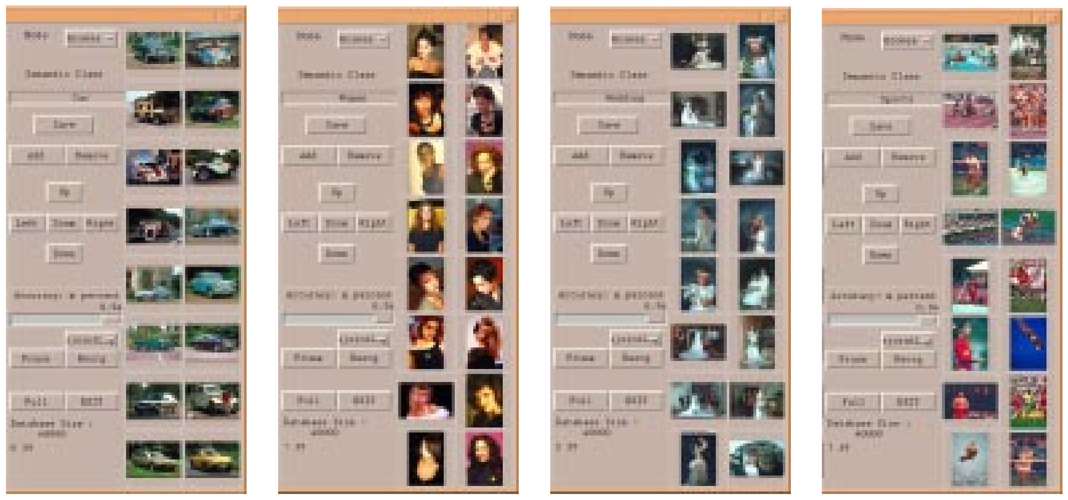


Fig. 4.3. The relevance sets/semantic classes of four manually chosen class: Car, Women, Wedding, and Sport.

will call the relevance set. The images in the relevance set are selected by the user as they move through the pyramid. The user can incrementally add or remove images from the relevance set at any time during the browsing process. For browsing, the user's objective may be to locate all images from a desired class. In this case, the relevance set may contain dissimilar groupings of images that represent the variations that can occur among images in the class. As the user browses through the database, the relevance set also becomes a buffer or clip board which stores all the images of interest to the user.

Once users have defined a set of relevant images, they may associate the relevance set with a semantic label. These relevance sets may then be stored and recalled based on the semantic label. Figure 4.3 shows four examples of relevance sets, corresponding to the semantic labels “car”, “women”, “wedding”, and “sport”. In essence, the relevance sets provide a simple mechanism for users to define abstract semantic classes.

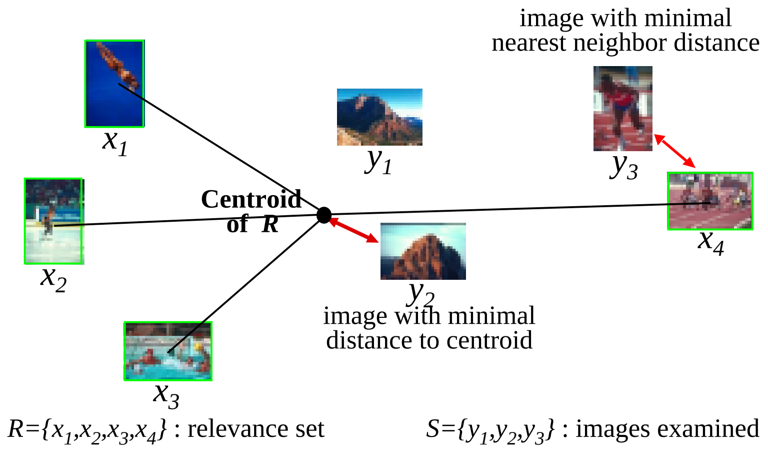


Fig. 4.4. An illustration of nearest neighbor distance.

4.3 Pruning

The objective of pruning is to remove images from the database that are not likely to be of interest to the user while retaining all or most potentially desirable images. This is useful since the reduced size of the pruned database makes it easier and more effective to browse.

While traditional queries methods attempt to find images that are likely to be of interest, pruning retains all images which are of possible interest, but at the expense of retaining many questionable images. Intuitively, pruning attempts to achieve high recall, but at the expense of precision; whereas traditional queries tend to emphasize precision over recall. The reduced precision of pruning is acceptable because the similarity pyramid structure allows the user to efficiently browse the resulting images.

Let $R = \{x_1, \cdots, x_M\}$ be a set of M relevant images that a user selects to be typical of images of interest. Then we would like to retain all images y in the database such that

$$d(y, R) \leq T$$

where $d(y, R)$ is the distance between y and the set of relevant images R, and T is a threshold.

Conventional relevance feedback approaches [45] define $d(y, R)$ to be the distance between y and the weighted average of relevance set as shown in Figure 4.4. However, this method fails when the set of desired images forms disjoint clusters in the feature space. Figure 4.4 illustrates the problem for the case of four relevant images denoted by $x_1, \cdots, x_4$. In this example, the centroid of the four sports images is near to the two undesired mountain images. While the desired sports image, y_3, is close to one example image, it is far from the weighted average.

To address this problem, we will use the nearest neighbor distance function

$$d(y, R) = \min_{i=1,\cdots,M} d(y, x_i) \tag{4.1}$$

where $d(y, x_i)$ is a simple histogram based distance function that incorporates color, edge and texture features [34, 44]. The nearest neighbor distance function works well when the set of desired images forms disjoint clusters in the feature space. In this way, the relevance set may represent a diverse set of images that are primarily related through a semantic label.

Our objective is then to select a threshold T that will have a high probability of retaining desired images, but will substantially reduce the size of the database. Stated more formally, let S be the set of desired images. Then define the accuracy of the pruning to be

$$\text{Accuracy} = P\{d(y, R) \leq T | y \in S\} \ .$$

Here accuracy has the same meaning as recall in the search-by-query problem, but we use the word accuracy because it has a more intuitive meaning in the application of pruning.

The problem is then to choose T in a way that achieves a fix level of pruning accuracy. To do this, we use a cross-validation strategy. Let $R_i = R - \{x_i\}$ be the set of relevance images with the image x_i removed. Then we define

$$\begin{aligned} d_i &= d(x_i, R_i) \\ \bar{d}_y &= d(y, R_1) \\ d_y &= d(y, R) \end{aligned}$$

where $y \in S$ is a desired image. Furthermore, we will denote the order statistics of d_i as $d_{(i)}$; so that $d_{(i)} \leq d_{(i+1)}$. The basic assumption of our analysis will be that the random variables d_i and $\bar{d}_y$ are independent and identically distributed. Under this assumption, it can be shown using standard methods from order statistics that (see Appendix C)

$$P\{\bar{d}_y \leq d_{(n)}\} = \frac{n}{M+1} \ .$$

Since we are using a nearest neighbor distance, the inequality $d_y \leq \bar{d}_y$ always holds. Therefore,

$$P\{d_y \leq d_{(n)}\} \geq \frac{n}{M+1} \ . \tag{4.2}$$

From (4.2), we see that if the pruning threshold is set to $T = d_{(n)}$ then the pruned database will contain on average $\frac{n}{M+1}$ of all the desired images. So for example, if the relevance set contains $M = 16$ images, and $d_{(16)}$ is used as the threshold, then on average $\frac{16}{17}$ or 94% of the desired images will be retained in the pruning. Importantly, this result does not depend on the statistics of database or the properties of the metric $d(y, x)$.

4.4 Reorganization

In addition to pruning, relevance feedback can be used to reorganize the database so the images of interest are grouped together. As with pruning, this reorganization makes the browsing process more efficient. Since the desired images often form disjoint clusters that are spread through the feature space, they will often be spread into disjoint groupings in the similarity pyramid. The reorganization step helps to move these desired images near one another in the similarity pyramid structure. While reorganization may be performed at any point in the browsing process, it is typically performed after pruning.

The following sections describe two alternative methods for reorganizing the similarity pyramid. Both methods depend on computation of a new dissimilarity metric based on the images in the relevance set.

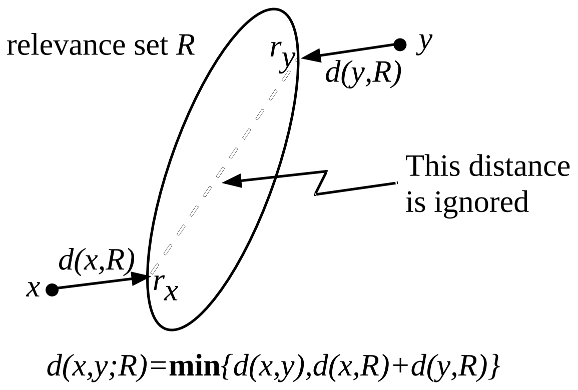


Fig. 4.5. An illustration of the "worm hole" distance. The distance from r_x to r_y is ignored.

4.4.1 Optimized Distance Function

For this section, we will assume that the distance function $d_\theta(y,x)$ is parameterized by a vector θ. In this work, θ is a 9 component vector containing the weightings corresponding to the L, a and b components of the color, edge and texture histogram features [34, 44].

The first step to reorganization is to compute the parameter $\hat{\theta}$ which optimizes the performance of the distance function on the relevance set R. To do this, we define a cost function $C(\theta,R)$ which uses cross-validation to estimate how well the distance function clusters the images in the relevance set. The detailed definition of $C(\theta,R)$ is given in Appendix D. With this cost function, we then compute

$$\hat{\theta} = \arg\min_{\theta} C(\theta,R) \ . \tag{4.3}$$

The minimization of (4.3) is done using conjugate gradient optimization. The resulting distance function $d_{\hat{\theta}}(y,x)$ is then used to rebuild the similarity pyramid. Because this parameter θ is selected to minimize the cost function $C(\theta,R)$, the reorganized pyramid does a better job of clustering together desired images.

4.4.2 Worm Hole Distance

In this section, we propose an alternative method to optimize the organization of the similarity pyramid. Rather then using the linear transformation of Section 4.4.1, we focus on nonlinear transformations of the dissimilarity metric. Our objective is to nonlinearly warp the feature space so that images in or near the relevance set are near each other. This distorted distance function is then used to rebuild the similarity pyramid so that images similar to the relevance set are grouped together.

In order to warp the feature space, we will define a new distance metric which can be easily computed from the initial distance metric $d(x, y)$. The new distance metric $d(x, y; R)$ depends on the relevance set R and is defined by

$$d(x, y; R) = \min\{d(x, y), d(x, R) + d(y, R)\} \tag{4.4}$$

where $d(x, y)$ is the dissimilarity function between images x and y, and $d(x, R)$ is the nearest neighbor distance between x and R defined by

$$d(x, R) = \min_{y \in R} d(x, y) \ .$$

We call this new metric of (4.4) the “worm hole” distance because points in the set R are considered to be collocated in the space. Figure 4.5 illustrates how the new metric works. Notice that the shortest distance between two points can include a short cut or worm hole through the relevance set. Since the set R may not be connected in the feature space, points that were originally far apart may be close in the new distorted space. In addition, the distance between any two points in the relevance set is zero i.e. for all $r_1, r_2 \in R$, $d(r_1, r_2; R) = 0$.

Perhaps surprisingly, we can show that the worm hole distance is a metric whenever the original distance $d(x, y)$ is a metric. This is a very useful property because it means that the worm hole distance can be used with fast search and clustering methods that depend on the use of the triangle inequality [33, 34, 44]. However, in order to prove this property, we must treat all the points in the relevance set R as an equivalence class represented by a single point.

More specifically, let Ω be a space, and let $R \subset \Omega$ be a set of points. Define $\Omega_R = (\Omega - R) \cup \{r_o\}$ where $r_o \in R$. Then Ω_R is a smaller space in which the points of R are treated as an equivalence class. Using this dissimilarity the following theorem is proven in Appendix E.

Theorem 4.4.1 *Let $d(x, y)$ be a metric on the space Ω and let $R \subset \Omega$ be a finite set of points in the space. Then $d(x, y; R) = \min\{d(x, y), d(x, R) + d(y, R)\}$ is a metric on the space Ω_R.*

In some cases, it may be useful to use more than one relevance set to distort the feature space. Let $d(x, y; R_1, \cdots, R_K)$ be a worm hole distance based on K relevance sets R_1 though R_K. The we define this distance recursively using the equation

$$d(x, y; R_1, \cdots, R_K) = \min \left\{ \begin{array}{l} d(x, y; R_1, \cdots, R_{K-1}), \\ d(x, R_K; R_1, \cdots, R_{K-1}) + d(y, R_K; R_1, \cdots, R_{K-1}) \end{array} \right\}$$

where we use the notation $d(x, R; R_1, \cdots, R_{K-1})$ to mean

$$d(x, R; R_1, \cdots, R_{K-1}) = \min_{y \in R} d(x, y; R_1, \cdots, R_{K-1}) \ .$$

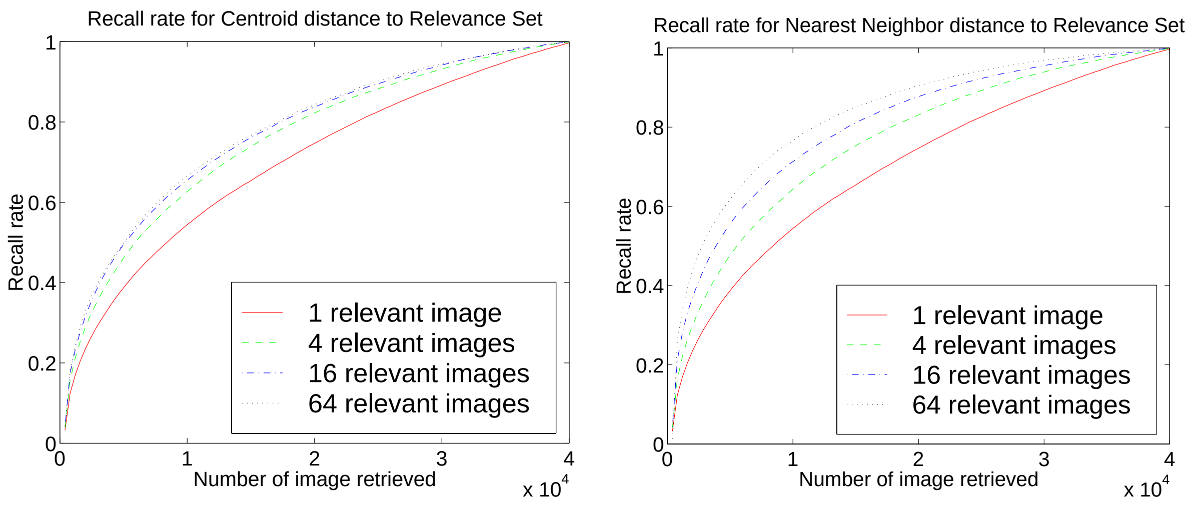
In Appendix E it is shown that this new distance function is also a metric. Furthermore, it is shown in [58] that the metric $d(x, y; R_1, \cdots, R_K)$ does not depend on the ordering of the sets $R_1, \cdots, R_K$.

Theorem 4.4.2 *Let $d(x, y)$ be a metric on the space Ω and let $R_k \subset \Omega$ be a finite set of points in the space for $k = 1, \cdots, K$. Let $\pi(1), \cdots, \pi(K)$ be a permutation of $1, \cdots, K$,*

$$d(x, y; R_1, \cdots, R_K) = d(x, y; R_{\pi(1)}, \cdots, R_{\pi(K)})$$

4.5 Experimental Results

We use a Corel Photo CD database consisting of 40,000 images which are evenly divided among 400 titles or classes. In this work, we assume each title to be a semantic class. For all experiments, the relevance sets were formed by randomly selecting images from each class. The effectiveness of the pruning and reorganization operations was then measured using the remaining images in each class. For each experiment, the images in the relevance set were excluded when computing performance estimates, so as not to unfairly bias comparisons.



(a) (b)

Fig. 4.6. The recall accuracy for (a) centroid distance, and (b) nearest neighbor distance. The performance of nearest neighbor distance improves with the increasing size of the relevance set.

Figure 4.6 compares the recall performance for nearest neighbor distance to the relevance set, and the distance to the centroid (i.e. mean feature vector) of the relevance set. The experiment was performed by averaging over all classes using relevance sets with sizes of 1, 4, 16, and 64 images. The curves corresponding to relevance sets of size 1 are the same for the nearest centroid and nearest neighbor distances, so in each plot this case serves as a baseline of performance. Notice that nearest neighbor distance not only performs uniformly better than the nearest centroid distance method, but it continues improving with increasing size of the relevance set. This is because these largely semantic classes tend to contain images which form disjoint groupings in the feature space.

Figure 4.7 illustrates the theoretic and experimental pruning accuracy (recall) when using the order statistic threshold described in Section 4.3. This experiment is performed by averaging over all classes and with relevance sets of size 16. Notice that the experimental curve is very consistent with the theoretical lower bound predicted

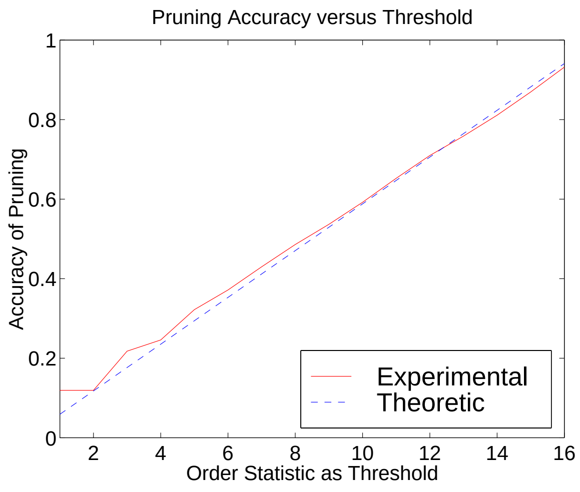


Fig. 4.7. The theoretic and experimental accuracy versus the rank of the order statistic used for pruning. All experiments use a relevance set of size 16.

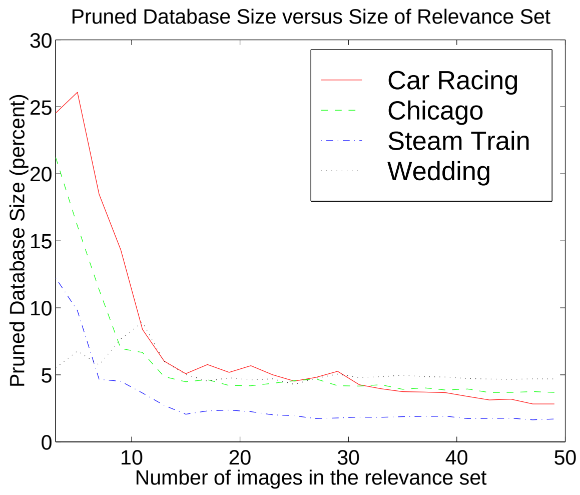


Fig. 4.8. The mean of the pruned database size versus the number of images in the relevance set. All examples use a pruning accuracy (recall) of 50%.

by (4.2).

Figure 4.8 illustrates the mean of the pruned database size with theoretical accuracy = 50% for four typical classes: *Car Racing, Chicago, Steam Train, Wedding.* Each plot was formed by averaging 10 random experiments. Notice that the size of the pruned database generally decreases with the size of the relevance set, but it decreases more rapidly for some classes than others. For example, classes such as "car racing" tend to be more semantically related, so a larger relevance set is useful. Alternatively, classes such as "wedding" only require a small number of images in the relevance set to accurately model the distribution in the feature space.

The next experiment compared the quality of pyramid organization using the original, optimized, and worm hole distance functions. In each case, a relevance set of 16 images was randomly selected from a class of images, and then the quality of organization for the remaining images in the class was measured. More specifically, we computed the dispersion which we define in Appendix F to be the normalized average distance in the pyramid between the M remaining images in the class that are closest in the pyramid. Improved organization should therefore result in a reduced

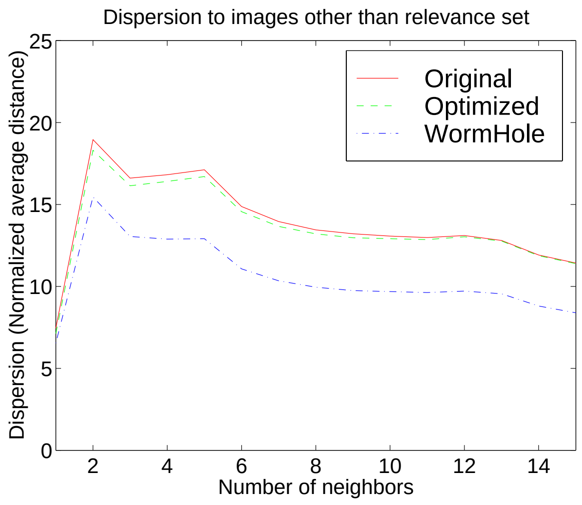


Fig. 4.9. The dispersion before and after reorganization: The dispersion from images in the relevance set to images in the same class but not in relevance set.

value of the dispersion.

Figure 4.9 shows the dispersion as a function of M averaged over all 400 classes. Notice that the optimized distance function described in Section 4.4.1 only slightly reduces the measured dispersion. However, the worm hole distance of Section 4.4.2 substantially reduces the dispersion. This is because the worm hole distance aggressively warps the space to move together images from the same class.

Figure 4.10 shows the results of reorganization of the similarity pyramid using the worm hole distance. The black rectangle is drawn around the 16 images contained in the relevance set R. Notice that the images near the relevance set match both the semantic and visual content of the relevance images quite closely. Interestingly, this happens even though the members of the relevance set differ substantially in both color and texture.

4.6 Conclusion

In this chapter, we propose methods for incorporating relevance feedback into the browsing process. Our approach is to allow the user to select images as they move through the database and copy them to a clipboard. We call this collection of images

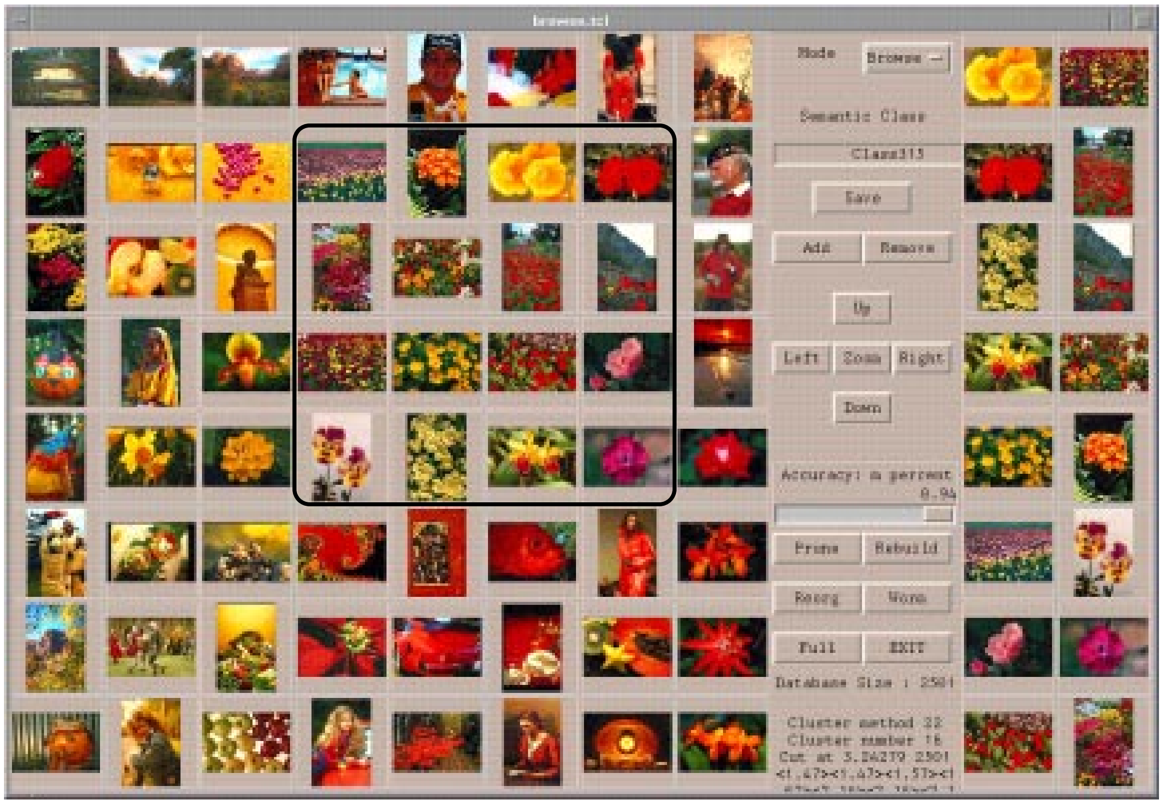


Fig. 4.10. A result of reorganization with worm hole distance. The images inside rectangle are members of relevance set.

a relevance set, and use it to both prune and reorganize the database. Pruning is done using a nearest neighbor distance and a threshold based on cross-validation. We propose two methods for reorganization. Both methods depend on the rebuilding of the pyramid with a new, more relevant distance function. The first method optimizes the distance metric to best separate the relevant images from the remaining images in the database. The second method is based on a "worm hole" distance which groups images of the relevance set together. Experiments indicate that the worm hole distance is quite effective at grouping images of interest together.

LIST OF REFERENCES

[1] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image content: The QBIC system. *IEEE Computer*, 28:23–31, September 1995.

[2] HongJiang Zhang and Di 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*, volume 2420, pages 36–46, San Jose, CA, February 9-10 1995.

[3] John MacCuish, Allen McPherson, Julio Barros, and Patrick Kelly. Interactive layout mechanisms for image database retrieval. In *Proc. of SPIE/IS&T Conf. on Visual Data Exploration and Analysis III*, volume 2656, pages 104–115, San Jose, CA, Jan 31-Feb 2 1996.

[4] Yossi Rubner, Leonidas Guibas, and Carlo Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *Proceedings of the ARPA Image Understanding Workshop*, May 1997.

[5] Scott Craver, Boon-Lock Yeo, and Minerva M. Yeung. Image browsing using data structure based on multiple space-filling curves. In *Proceedings of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 1-4 1998.

[6] Yihong Gong, Hongjiang Zhang, and Chua Hock Chuan. An image database system with fast image indexing capability based on color histograms. In *Proceedings of 1994 IEEE Region 10's ninth annual international conference : Frontiers of computer technology*, pages 407–411, Singapore, August 22-26 1994.

[7] HongJiang Zhang, Yihong Gong, Chien Y. Low, and Stephen W. Smoliar. Image retrieval based on color features: An evaluation study. In *Proc. of SPIE/IS&T Conf. on Storage and Archiving Systems*, volume 2606, pages 212–220, Philadelphia, PA, October 25-26 1996.

[8] Charles E. Jacobs, Adams Finkelstein, and David H. Salesin. Fast multiresolution image querying. In *ACM Siggraph 95 proceedings*, pages 277–285, Los Angeles, CA, August 9-11 1995.

[9] Srinivas Sista, Charles A. Bouman, and Jan P. Allebach. Fast image search using a multiscale stochastic model. In *Proc. of IEEE Int'l Conf. on Image Proc.*, pages 225–228, Washington, DC, October 23-26 1995.

[10] Jau-Yuen Chen, Charles A. Bouman, and Jan P. Allebach. Stochastic models for fast multiscale image search. In *Joint Conference of Classification Society of North America and Numerical Taxonomy Group*, Amherst, MA, June 13-16 1996.

[11] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[12] Michael J. Swain and Dana H. Ballard. Color indexing. *Intern. Journal of Computer Vision*, 7(1):11–32, 1991.

[13] James Hafner, Harpreet S. Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(7):729–736, July 1995.

[14] Markus Stricker and Markus Orengo. Similarity of color images. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases III*, volume 2420, pages 381–392, San Jose, CA, February 9-10 1995.

[15] Shunsuke Ihara. *Information Theory for continuous systems.* World Scientific, Singapore, 1993.

[16] Ullas Gargi, Scott Oswald, David Kosiba, Sadashiva Devadiga, and Rangachar Kasturi. Evalutation of video sequence indexing and hierarchical video indexing. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases III*, volume 2420, pages 144–151, San Jose, CA, February 9-10 1995.

[17] Thomas Frese, Charles A. Bouman, and Jan P. Allebach. Methodology for designing image similarity metrics based on human visual system models. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases II*, pages 10–13, February 1997.

[18] Minerva M. Yeung and Bede Liu. Efficient matching and clustering of video shots. In *Proc. of IEEE Int'l Conf. on Image Proc.*, volume I, pages 338–341, Washington, DC, October 23-26 1995.

[19] Minerva M. Yeung and Boon-Lock Yeo. Video visualization for compact presentation and fast browsing of pictorial content. *IEEE Trans. on Circuits and Systems for Video Technology*, 7(5):771–785, October 1997.

[20] Ruggero Milanese, David Squire, and Thierry Pun. Correspondence analysis and hierarchical indexing for content-based image retrieval. In *Proc. of IEEE Int'l Conf. on Image Proc.*, volume 3, pages 859–862, Lausanne Switzerland, September 16-19 1996.

[21] M. T. Orchard. A fast nearest-neighbor search algorithm. In *Proc. of IEEE Int'l Conf. on Acoust., Speech and Sig. Proc.*, pages 2297–2300, Toronto, Canada, May 1991.

[22] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, April 1973.

[23] V. Ruiz. An algorithm for finding nearest neighbors in (approximately) constant time. *Pattern Recognition Letters*, 4:145–157, July 1986.

[24] 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. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases IV*, volume 3022, pages 392–403, San Jose, CA, February 1-2 1996.

[25] A. Berman and L. Shapiro. Efficient image retrieval with multiple distance measures. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases V*, volume 3022, San Jose, CA, February 13-14 1997.

[26] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[27] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases IV*, volume 2670, pages 62–73, San Jose, CA, February 1-2 1996.

[28] R. Ng and A. Sedighian. Evaluating multi-dimensional indexing structure for images transformed by principal component analysis. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases IV*, volume 2670, pages 50–61, San Jose, CA, February 1-2 1996.

[29] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. on Computers*, 24(7):750–753, July 1975.

[30] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 71–79, San Jose, CA, June 1995.

[31] D. A. White and R. Jain. Similarity indexing with the S-tree. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases IV*, volume 2670, pages 62–73, San Jose, CA, February 1-2 1996.

[32] 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. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases V*, volume 3022, pages 110–120, San Jose, CA, February 13-14 1997.

[33] Jau-Yuen Chen, Charles A. Bouman, and Jan P. Allebach. Fast image database search using tree-structured VQ. In *Proc. of IEEE Int'l Conf. on Image Proc.*, volume 2, pages 827–830, Santa Barbara, CA, October 26-29 1997.

[34] Jau-Yuen Chen, Charles A. Bouman, and John Dalton. Similarity pyramids for browsing and organization of large image databases. In *Proc. of SPIE/IS&I Conf. on Human Vision and Electronic Imaging III*, volume 3299, San Jose, CA, January 26-29 1998.

[35] G. N. Lance and W.T. Williams. A general theory of classificatory sorting strategies. i. hierarchical systems. *Computer Journal*, 9:373–380, 5 1966.

[36] P.H.A. Sneath and R.R. Sokal, editors. *Numerical Taxonomy.* Freeman, San Francisco, 1973.

[37] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical, Statistics and Probability*, volume 1, pages 281–297, 1964.

[38] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Trans. on Communications*, COM-28:84–95, January 1980.

[39] Minerva M. Yeung, Boon-Lock Yeo, and Bede Liu. Extracting story units from long programs for video browsing and navigation. In *IEEE International Conference on Multimedia Computing and Systems*, pages 296–305, Hiroshima, Japan, June 17-21 1996.

[40] Anil K. Jain and Richard C. Dubes, editors. *Algorithms for Clustering Data.* Prentice Hall, New Jersey, 1988.

[41] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, editors. *Introduction to Algorithms.* McGraw-Hill Book Company, New York, 1990.

[42] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Sil verman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *Proc. of the Fifth Annual ACM-SIAM Symposium on Discrete Algorith ms*, pages 573–582, 1994.

[43] Jau-Yuen Chen, Charles A. Bouman, and Jan P. Allebach. Multiscale branch and bound image database search. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases V*, volume 3022, pages 133–144, San Jose, CA, February 13-14 1997.

[44] Jau-Yuen Chen, Charles A. Bouman, and John Dalton. Hierarchical browsing and search of large image databases. *Submitted to IEEE Trans. on Image Processing*, 1998. 1998.

[45] Gerard Salton. *The SMART Retrieval System.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971.

[46] Thomas P. Minka and Rosalind W. Picard. Interactive learning using a "society of models". Technical Report 349, MIT Media Laboratory, 1995.

[47] Rosalind W. Picard. A society of models for video and image libraries. Technical Report 360, MIT Media Laboratory, 1996.

[48] Ingemar J. Cox, Matt L. Miller, Stephen M. Omohundro, and Peter N. Yianilos. Pichunter: Bayesian relevance feedback for image retrieval. In *Proceedings of International Conference on Pattern Recognition*, volume 3, pages 361–369, Vienna, Austria, August 1996.

[49] Leonid Taycher, Marco La Cascia, and Stan Sclaroff. Image digestion and relevance feedback in the Imagerover WWW search engine. In *Proceedings of International Conference on Visual Information*, San Diego, CA, December 15-17 1997.

[50] Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Relevance feedback techniques in interactive content-based image retrieval. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases VI*, volume 3312, pages 25–36, San Jose, CA, January 26-29 1998.

[51] Simone Santini and Ramesh Jain. Interfaces for emergent semantics in multimedia databases. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases VII*, volume 3656, pages 167–175, San Jose, CA, January 26-29 1999.

[52] Matthew E. J. Wood, Neill W. Campbell, and Bary T. Thomas. Iterative refinement by relevance feedback in content-based. In *ACM Multimedia 98*, pages 13–20, Bristol, UK, September 1998.

[53] Ana B. Benitez, Mandis Beigi, and Shih-Fu Chang. A content-based image meta-search engine using relevance feedback. *IEEE Internet Computing Magazine*, 2(4):59–69, July 1998.

[54] J-Y. Chen, C. Taşkiran, E. J. Delp, and C. A. Bouman. ViBE: A new paradigm for video database browsing and search. In *Proc. of IEEE Workshop on Content-Based Image and Video Databases*, pages 96–100, Santa Barbara, CA, June 21 1998.

[55] Cuneyt Taskiran, Jau-Yuen Chen, Charles A. Bouman, and Edward J. Delp. A compressed video database structured for active browsing and search. In *Proc. of IEEE Int'l Conf. on Image Proc.*, volume III, Chicago, IL, October 4-7 1998.

[56] Jau-Yuen Chen, Charles A. Bouman, and John Dalton. Active browsing using similarity pyramids. In *Proceedings of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2-4 1998.

[57] Jau-Yuen Chen, Charles A. Bouman, and John Dalton. Active browsing using similarity pyramids. In *Proc. of SPIE/IS&T Conf. on Storage and Retrieval for Image and Video Databases VII*, volume 3656, pages 144–154, San Jose, CA, January 26-29 1999.

[58] Jau-Yuen Chen, Charles A. Bouman, and John Dalton. *Active Browsing using Similarity Pyramids.* PhD thesis, Purdue University, West Lafayette, IN, May 1999.

[59] Teuvo Kohonen, editor. *Self-Organizing Maps.* Springer, Germany, 1995.

APPENDICES

Appendix A: 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[43], 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 $\frac{\pi}{8}$. An extra bin is reserved for the case of $\Theta = \emptyset$.

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

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

Appendix B: SOM Pyramid Construction

We use SOM [59] 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 $\lceil \frac{N}{64} \rceil$ 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.

Appendix C: Order Statistic

In this section, we compute $P\{\bar{d}_y \leq d_{(n)}\}$, the probability function we discussed in Section 4.3.

Assume $\bar{d}_y$ and $d_i, i = 1 \cdots M$ are i.i.d. with $P\{d_i \leq t\} = F(t)$ and $d_{(i)}$ is the i^{th} order statistic of $\{d_i\}_{i=1 \cdots M}$, with $P\{d_{(n)} \leq t\} = F_{d_{(n)}}(t)$.

It is well known that

$$dF_{d_{(n)}}(d_{(n)}) = \frac{M!}{(n-1)!(M-n)!} F(d_{(n)})^{n-1}(1 - F(d_{(n)})^{n-j} dF(d_{(n)})$$

Under these assumption, $P\{\bar{d}_y \leq d_{(n)}\}$ can be found by:

$$\begin{aligned} P\{\bar{d}_y \leq d_{(n)}\} &= E_{d_{(n)}}[P\{\bar{d}_y \leq d_{(n)} | d_{(n)}\}] \\ &= E_{d_{(n)}}[F(d_{(n)})] \end{aligned}$$

$$
\begin{aligned}
&= \int F(d_{(n)}) dF_{d_{(n)}}(d_{(n)}) \\
&= \int F(d_{(n)}) \frac{M!}{(n-1)!(M-n)!} F(d_{(n)})^{n-1}(1-F(d_{(n)})^{n-j} dF(d_{(n)}) \\
&\quad \text{(But } y = F(X) \text{ is a uniform distribution on [0, 1])} \\
&= \int_0^1 t \frac{M!}{(n-1)!(M-n)!} t^{n-1}(1-t)^{n-j} dt \\
&\quad \text{(This is the expectation of beta(n, n-j+1) distribution)} \\
&= \frac{n}{M+1}
\end{aligned}
$$

Appendix D: Cost function

In this appendix, we define the cost function $C(\theta, R)$ which uses cross-validation to estimate how well the distance function clusters the images in the relevance set. This cost function will be used to compute the parameter $\hat{\theta}$ which optimizes the performance of the distance function on the relevance set R.

To define $C(\theta, R)$, we first randomly select K images out of the pruned database to form the set $Y = \{y_i, \cdots, y_K\}$. Then define $d_{ij} = d_\theta(y_j, R_i)$ and let $d_{i(j)}$ be the j^{th} order statistic of $\{d_{ij}\}_{j=1\cdots K}$, so that $d_{i(j)} \leq d_{i(j+1)}$. For the boundary case, further assume that $d_{i(0)} = 0$. Recalling that $d_i = d(x_i, R_i)$, then we define π_i as the rank of d_i such that $d_{i(\pi_i)} \leq d_i \leq d_{i(\pi_i+1)}$. Using these definitions, we define the continuously valued rank of d_i as

$$
r_i = \pi_i + \frac{d_i - d_{i(\pi_i)}}{d_{i(\pi_i+1)} - d_{i(\pi_i)}}. \tag{D.1}
$$

Notice that (D.1) is chosen so that r_i is a continuous function of d_i. This is important since it facilitates optimization. The total cost function is then given by

$$
C(\theta, R) = \sum_{i=1\cdots M} r_i \tag{D.2}
$$

Appendix E: Worm Hole Distance

In this appendix, we show that the worm hole distance introduced in Section 4.4.2 is a distance metric. Let Ω be a space, let $R \subset \Omega$, and let Ω_R be a smaller space in which the points of R are treated as an equivalence class.

Theorem E.1 (4.4.1) *Let $d(x, y)$ be a metric on the space Ω and let $R \subset \Omega$ be a* *finite set of points in the space. Then* $d(x,y;R) = \min\{d(x,y), d(x,R) + d(y,R)\}$ *is a metric on the space* Ω_R*.*

proof:

We need to show that $d(x,y;R) \geq 0$,

- $\forall x \neq y \in \Omega_R$, $d(x,y;R) > 0$
- $\forall x \in \Omega_R$, $d(x,x;R) = 0$
- $d(x,y;R) = d(y,x;R)$
- $d(x,y;R) + d(y,z;R) \geq d(x,z;R)$

The first three properties are easily verified. However, we need to verify that the triangle inequality holds. First notice that

$$d(x,y;R)+d(y,z;R) = \min\{\underbrace{d(x,y)}_{(term1)}, \underbrace{d(x,R)+d(y,R)}_{(term2)}\}+\min\{\underbrace{d(y,z)}_{(term3)}, \underbrace{d(y,R)+d(z,R)}_{(term4)}\} \tag{E.1}$$

There are four cases for the two *min* functions in (E.1). For each case, we will show that $d(x,z;R)$ is a lower bound.

Case 1: Assume the minima are achieved by terms 1 and 3.

$$\begin{aligned} d(x,y;R) + d(y,z;R) &= d(x,y) + d(y,z) \\ &\geq d(x,z) \\ &\geq \min\{d(x,z), d(x,R) + d(z,R)\} \\ &= d(x,z;R) \end{aligned}$$

where the first inequality is simply the triangle inequality of metric $d(x,y)$ and the second inequality is the result of *min* operator.

Case 2: Assume the minima are achieved by terms 2 and 4.

$$\begin{aligned} d(x,y;R) + d(y,z;R) &= d(x,R) + d(y,R) + d(y,R) + d(z,R) \\ &\geq d(x,R) + d(z,R) \\ &\geq \min\{d(x,z), d(x,R) + d(z,R)\} \\ &= d(x,z;R) \end{aligned}$$

Case 3: Assume the minima are achieved by terms 2 and 3.

Let r_x, r_y and r_z be chosen so that

$$\begin{aligned} d(x, r_x) &= \min_{t \in R} d(x, t) \\ d(y, r_y) &= \min_{t \in R} d(y, t) \\ d(z, r_z) &= \min_{t \in R} d(z, t) \ . \end{aligned}$$

Then we have that

$$\begin{aligned} d(x, y; R) + d(y, z; R) &= d(x, R) + d(y, R) + d(y, z) \\ &= d(x, r_x) + d(y, r_y) + d(y, z) \\ &\geq d(x, r_x) + d(r_y, z) \\ &\geq d(x, r_x) + d(r_z, z) \\ &= d(x, R) + d(z, R) \\ &\geq \min\{d(x, z), d(x, R) + d(z, R)\} \\ &= d(x, z; R) \ . \end{aligned}$$

Case 4: The minima are achieved by terms 1 and 4.

This proof is similar to case 3.

QED

We next show that the distance function $d(x, y; R_1, \cdots, R_K)$ is a distance metric on the space $\Omega_{R_1, \cdots, R_K}$. Here $\Omega_{R_1, \cdots, R_K}$ is a smaller version of the space Ω where each set of points R_k is treated as an equivalence class.

Theorem E.2 *Let $d(x, y)$ be a metric on the space Ω and let $R_k \subset \Omega$ be a finite set of points in the space for $k = 1, \cdots, K$. Then $d(x, y; R_1, \cdots, R_K)$ is a metric on the space $\Omega_{R_1, \cdots, R_K}$.*

proof:

We only need to verify that $d(x, y; R_1, \cdots, R_K)$ obeys the triangle inequality. The other properties are easily checked.

Our proof is by induction on K. We have already shown this property for $K = 1$. Assume it holds for $K - 1$, and define $\bar{d}(x, y) = d(x, y; R_1, \cdots, R_{K-1})$. Then $\bar{d}(x, y)$ is a metric, and $d(x, y; R_1, \cdots, R_K) = \bar{d}(x, y; R_K)$ must therefore be a metric by the previous theorem.

QED

We next show that the distance function $d(x, y; R_1, \cdots, R_K)$ is independent of the order of R_K. We first prove it for the case $K = 2$, and then show the general case of pairwise interchange. With the pairwise interchange property, the result is easily generalized to any permutation.

Lemma E.1 *Let $d(x, y)$ be a metric on the space Ω and let $R_1, R_2 \subset \Omega$ be finite sets of points in the space. Then $d(x, y; R_1, R_2) = d(x, y; R_2, R_1)$.*

proof:

Let r_{x1}, r_{y1}, r_{x2}, r_{y2}, r_1, and r_2 be chosen so that

$$\begin{aligned}
d(x, r_{x1}) &= \min_{t \in R_1} d(x, t) \\
d(y, r_{y1}) &= \min_{t \in R_1} d(y, t) \\
d(x, r_{x2}) &= \min_{t \in R_2} d(x, t) \\
d(y, r_{y2}) &= \min_{t \in R_2} d(y, t) \\
d(r_1, r_2) &= \min_{u \in R_1} \min_{v \in R_2} d(u, v) \ .
\end{aligned}$$

$$\begin{aligned}
d(x, y; R_1, R_2) &= \min\{d(x, y; R_1), d(x, R_2; R_1) + d(y, R_2; R_1)\} \\
&= \min \left\{ \begin{array}{l} \min\{d(x, y), d(x, R_1) + d(y, R_1)\}, \\ \min\limits_{t \in R2} d(x, t; R_1) + \min\limits_{t \in R2} d(y, t; R_1)\} \end{array} \right\} \\
&= \min \left\{ \begin{array}{l} \min\{d(x, y), d(x, r_{x1}) + d(y, r_{y1})\}, \\ \min\limits_{t \in R2} \min\{d(x, t), d(x, R_1) + d(t, R_1)\} + \\ \min\limits_{t \in R2} \min\{d(y, t), d(y, R_1) + d(t, R_1)\} \end{array} \right\}
\end{aligned}$$

The second equality is simply the replacement of r_{x1} and r_{y1} and the expansion of worm hole distance distance for $d(x, R_2; R_1)$ and $d(y, R_2; R_1)$. The third equality is followed by the definition of worm hole distance for $d(x, t; R_1)$ and $d(y, t; R_1)$. With the expansion of min operator, this equation can be rewritten as

$$\begin{aligned} d(x, y; R_1, R_2) &= \min \left\{ \begin{array}{l} d(x, y), \\ d(x, r_{x1}) + d(y, r_{y1}), \\ d(x, r_{x2}) + d(y, r_{y2}), \\ d(x, r_{x2}) + d(y, r_{y1}) + d(r_1, r_2), \\ d(x, r_{x1}) + d(r_1, r_2) + d(y, r_{y2}), \\ d(x, r_{x1}) + d(r_1, r_2) + d(y, r_{y1}) + d(r_1, r_2) \end{array} \right\} \\ &= \min \left\{ \begin{array}{l} d(x, y), \\ d(x, r_{x1}) + d(y, r_{y1}), \\ d(x, r_{x2}) + d(y, r_{y2}), \\ d(x, r_{x2}) + d(y, r_{y1}) + d(r_1, r_2), \\ d(x, r_{x1}) + d(r_1, r_2) + d(y, r_{y2}) \end{array} \right\} \end{aligned}$$

The second equality is followed by the nonnegative property of $d(r_1, r_2)$. Similarly, $d(x, y; R_2, R_1)$ can be expressed as:

$$d(x, y; R_2, R_1) = \min \left\{ \begin{array}{l} d(x, y), \\ d(x, r_{x2}) + d(y, r_{y2}), \\ d(x, r_{x1}) + d(y, r_{y1}), \\ d(x, r_{x1}) + d(y, r_{y2}) + d(r_1, r_2), \\ d(x, r_{x2}) + d(r_1, r_2) + d(y, r_{y1}) \end{array} \right\}$$

From above two equations, it is obvious that

$$d(x, y; R_1, R_2) = d(x, y; R_2, R_1)$$

QED

Lemma E.2 *Let $d(x, y)$ be a metric on the space Ω and let $R_k \subset \Omega$ be a finite set of points in the space for $k = 1, \cdots, K$. Then, for all $1 \leq m \leq K$,*

$$d(x, y; R_1, \cdots, R_m, R_{m+1}, \cdots, R_K) = d(x, y; R_1, \cdots, R_{m+1}, R_m, \cdots, R_K)$$

proof:

Let $d_{R_1,\cdots,R_{m-1}}(x,y) = d(x,y;R_1,\cdots,R_{m-1})$,

$$\begin{aligned} d(x,y;R_1,\cdots,R_{m-1},R_m,R_{m+1}) &= d_{R_1,\cdots,R_{m-1}}(x,y;R_m,R_{m+1}) \\ &= d_{R_1,\cdots,R_{m-1}}(x,y;R_{m+1},R_m) \\ &= d(x,y;R_1,\cdots,R_{m-1},R_{m+1},R_m) \end{aligned}$$

We can also rewrite this as $d_{R_1,\cdots,R_{m-1},R_m,R_{m+1}}(x,y) = d_{R_1,\cdots,R_{m-1},R_{m+1},R_m}(x,y)$

$$\begin{aligned} & d(x,y;R_1,\cdots,R_{m-1},R_m,R_{m+1},R_{m+2},\cdots,R_K) \\ = \; & d_{R_1,\cdots,R_{m-1},R_m,R_{m+1}}(x,y;R_{m+2},\cdots,R_K) \\ = \; & d_{R_1,\cdots,R_{m-1},R_{m+1},R_m}(x,y;R_{m+2},\cdots,R_K) \\ = \; & d(x,y;R_1,\cdots,R_{m-1},R_{m+1},R_m,,R_{m+2},\cdots,R_K) \end{aligned}$$

QED

Lemma E.3 *Let $d(x,y)$ be a metric on the space Ω and let $R_k \subset \Omega$ be a finite set of points in the space for $k = 1,\cdots,K$. Then, for all $1 \leq m < n \leq K$,*

$$d(x,y;R_1,\cdots,R_m,\cdots,R_n,\cdots,R_K) = d(x,y;R_1,\cdots,R_n,\cdots,R_m,\cdots,R_K)$$

proof:

This is just a simple application of Lemma E.2. We first continuously swap adjacent relevance sets to move R_m to the place it should be, and then continuously swap adjacent relevance set to move R_n to the place it should be.
QED

Theorem E.3 (4.4.2) *Let $d(x,y)$ be a metric on the space Ω and let $R_k \subset \Omega$ be a finite set of points in the space for $k = 1,\cdots,K$. Let $\pi(1),\cdots,\pi(K)$ be a permutation of $1,\cdots,K$,*

$$d(x,y;R_1,\cdots,R_K) = d(x,y;R_{\pi(1)},\cdots,R_{\pi(K)})$$

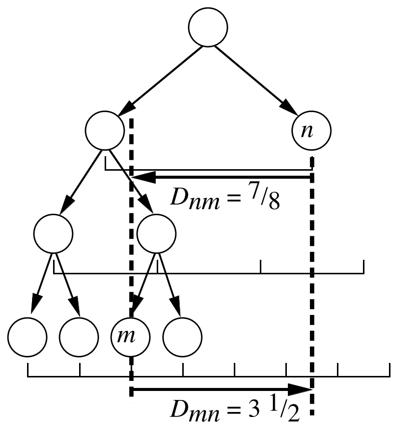


Fig. F.1. This figure illustrates the distance measured between two images in the similarity pyramid. The distance is an asymmetric function because it is measured relative to the first image.

proof:

This can be proved by construction. Starting from R_1, we can find an n such that $\pi(n) = 1$, and apply Lemma E.3 to swap R_1 and $R_{\pi(1)}$. We then continue this construction, until the position of every R_n in $R_1, \cdots, R_K$ is corrected.

QED

Appendix F: Dispersion for Reorganization

In this section, we will modify the *dispersion* measures proposed in [34, 44] for evaluating the quality of pyramid organization. The dispersion measures the average normalized distance between similar images in the pyramid. Figure F.1 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.

The dispersion measures the normalized distance between image n, and the M image in the same class that are closest to the image n in the pyramid. More formally, let $\pi(n, m)$ denotes the index of the m^{th} closest image in the class, so that $D_{n\pi(n,m-1)} \leq D_{n\pi(n,m)} \leq D_{n\pi(n,m+1)}$. Then the dispersion for image n is given by

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

where $\underline{D}(M)$ is a normalizing constant equal to minimum average distance for an ideal packing of M images. $\underline{D}(M)$ is computed by:

$$\begin{aligned} L(m) &= \min_k \{k : 2k(k+1) \geq m\} \\ \underline{D}(M) &= \frac{1}{M} \sum_{m=1}^{M} L(m) \end{aligned}$$

VITA

Jau-Yuen Chen was born in Pingtong, Taiwan, ROC on January 25, 1963. He received BS degree in electrical engineering from National Taiwan University in 1985. and a master degree in electrical engineering from National Taiwan University in 1987. From 1987 to 1994, he was an assistant scientist at the Chung-Shan Institute of Science and Technology in Taiwan, ROC He received his PhD degree in electrical and computer engineering from Purdue University in 1999.