

Adaptively browsing image databases with PIBE

Ilaria Bartolini · Paolo Ciaccia · Marco Patella

Published online: 30 September 2006
© Springer Science + Business Media, LLC 2006

Abstract Browsing large image collections is a complex and often tedious task, due to the semantic gap existing between the user subjective notion of similarity and the one according to which a browsing system organizes the images. In this paper we propose PIBE, an *adaptive* image browsing system, which provides users with a hierarchical view of images (the Browsing Tree) that can be customized according to user preferences. A key feature of PIBE is that it maintains local similarity criteria for each portion of the Browsing Tree. This makes it possible both to avoid costly global reorganization upon execution of user actions and, combined with a persistent storage of the Browsing Tree, to efficiently support multiple browsing tasks. We present the basic principles of PIBE and report experimental results showing the effectiveness of its browsing and personalization functionalities.

Keywords Image databases · Browsing · Personalization · Similarity criteria

1 Introduction

Browsing large image collections is a complex and often tedious task. Most of the commercial image browsers are not able to provide effective and efficient techniques for exploring very large databases, since they usually show images in a sequential way [5]. On the other hand, some recently proposed browsing systems [4, 6, 8, 13, 15] organize images in a hierarchical structure that groups together similar images based

I. Bartolini (✉) · P. Ciaccia · M. Patella
DEIS, University of Bologna, Italy
e-mail: ibartolini@deis.unibo.it

P. Ciaccia
e-mail: pciaccia@deis.unibo.it

M. Patella
e-mail: mpatella@deis.unibo.it

on their low-level features (e.g., color, texture). This is helpful for users, because they can efficiently browse the whole collections looking for images similar to the goal one(s). Any system-based organization, however, is prone to be considered inaccurate by users, who typically have a perceived notion of image similarity that differs from the one provided by the system (this is called *semantic gap* in [14]). Moreover, even the user notion of similarity may vary across different browsing sessions, depending on the actual browsing task.

It is our opinion that an effective system should give the user the opportunity of *personalizing* the default browsing structure, so as to make it closer to user's current preferences; moreover, in order to improve the system usability over multiple browsing sessions (possibly with similar goals), such personalized structure should be made *persistent*. Such two requirements already led to efficient techniques for query-based retrieval systems [2], but have so far almost being ignored by browsing systems. Indeed, state-of-the-art browsing systems suffer one or more of the following major limitations (see also Section 7):

Static browsing structure: The (usually hierarchical) organization of images cannot be altered by the user. This implies that no personalization at all is possible to match user preferences.

No persistence of personalization: This means that, even if the browsing system allows the user to personalize the organization of images, this is limited to the specific browsing session. Thus, each new session starts with the original (default) organization.

Global similarity criterion: Browsing systems that allow the user to modify the browsing structure through interaction typically do so by changing the *global* similarity criterion according to which images are organized. In doing so, the *whole* DB or, at least, a large part of it needs to be taken into account, which is clearly unfeasible with large image collections.

Motivated by above observations, in this paper we propose PIBE (*Personalizable Image Browsing Engine*), an adaptive image browsing system, which aims to provide users with an intuitive, easy-to-use, structured view of the images in the DB and complements it with ideas from the field of adaptable content-based similarity search.¹ In particular, PIBE provides the user with a customizable hierarchical browsing structure (called the *Browsing Tree*), whose changes persist across different sessions, and with a set of graphical *personalization actions* that can be used to modify the Browsing Tree. The effects of such actions are defined so as to guarantee that only a “local” reorganization of the image DB is required. This is possible since PIBE maintains specific similarity criteria for each portion (sub-tree) of the Browsing Tree. Note that using local similarity criteria is not only beneficial from an efficiency point of view, but it is also the key to reconcile persistence and support of multiple browsing tasks.

¹With respect to [1], this paper: (1) describes in more details the Browsing Tree, (2) provides a new 3-D visualization modality, (3) includes four new personalization actions, and (4) new experiments demonstrating the usability of PIBE.

The rest of the paper is organized as follows. Section 2 describes PIBE architecture and discusses its basic design principles. Section 3 provides details on the Browsing Tree. Browsing and personalization facilities are described in Sections 4 and 5, respectively. In Section 6, we present experimental results showing the effectiveness of PIBE. Section 7 surveys the state-of-the-art and Section 8 concludes.

2 PIBE architecture and principles

Figure 1 illustrates the PIBE architecture. An *initial* Browsing Tree (BT) is first automatically built over the image features. At run-time the user interacts with the system through a GUI, which, among others, accepts specific requests to modify the BT structure. These are actually managed by the *Browsing Processor* component. The *Visualizer* implements 2-D and 3-D visualization techniques to display images arranged on the screen so as to respect their mutual dissimilarities.

The user interface includes a toolbar (see Fig. 2) to customize the current view of the image DB, by zooming in/out, increasing/decreasing the size of image thumbnails, and combining the zooming in with the increment of image size. Two buttons permit to show/hide additional information related to each BT node, e.g., node identifier, name of the representative image, number of images in the node (such data are useful for experimental purposes). Other three buttons help the user in traversing the BT, by navigating backward (to the parent node) and forward (to the last visited child of the current node), and to return to the root node of the BT. Finally, the “3-D” button enables the view of images in a 3-D *viewing room*. Controls are provided to adapt the room projection, by rotating and/or zooming into it, and to choose whether images are to be displayed as “cubes” or as flat icons (see Fig. 5).

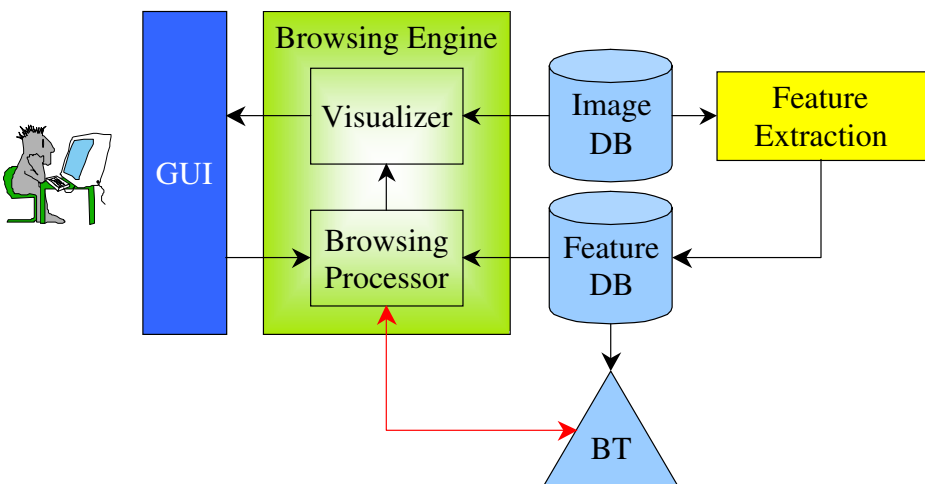


Fig. 1 The PIBE architecture

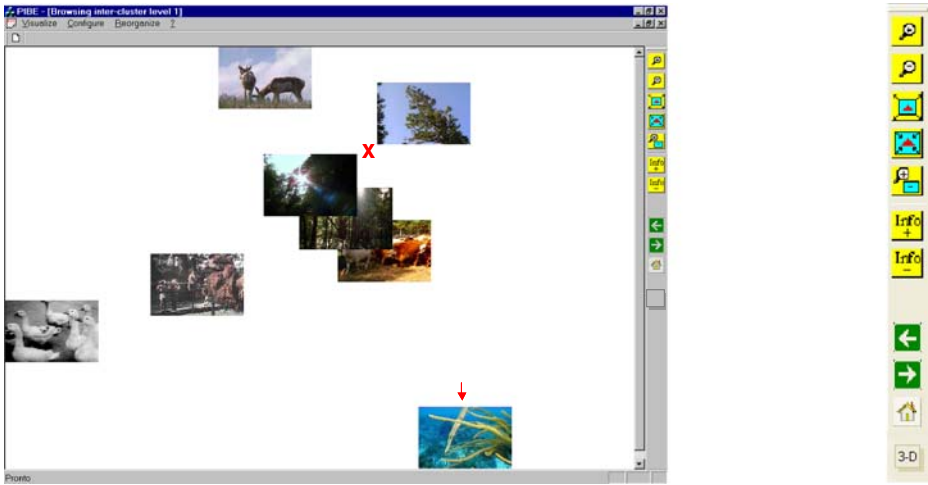


Fig. 2 The PIBE GUI. The zoomed toolbar is shown on the *right*

2.1 Principles and implementation choices

In designing PIBE we relied on the principle that an effective browsing structure can only be obtained through a synergic interaction between the user and the system. Thus, the user should be able to modify the default browsing structure, so that it better reflects her preferences concerning images' organization, and the system should smoothly adapt itself to such user actions. In PIBE, this is obtained through the interaction of four principal components, namely: personalization actions, image descriptors, (dis)similarity functions, and clustering algorithms.

Personalization actions: These allow the user to modify the current BT by moving images throughout the tree. In particular (see Section 5), nodes of the BT, corresponding to clusters of images, can be split, merged, and grafted so as to let the BT organization reflect user expectation. Each action has the effect of modifying the BT structure, e.g., by adding/removing children to/from a node, and the associated information stored therein.

Image visual descriptors: For PIBE to be able to automatically organize images and to smoothly upgrade the BT upon user actions, each image has to be represented by low-level features, such as color, texture, and shape, that can effectively describe image content. In PIBE, so-obtained visual descriptors are only required to be points (*feature vectors*) in a N -dimensional space. Usually, the more the descriptors are robust in representing the image content, the higher the dimensionality of the associated space and the complexity of the algorithms applied for their extraction and comparison increases. In the current version of our prototype, each image is represented in the HSV color space as a 32-D histogram, obtained by quantizing

the *Hue* and the *Saturation* components in eight and four equally-spaced intervals, respectively.

Classes of similarity functions: Comparing images is the basis for displaying them in a spatial layout, for clustering them (see below), and for searching images similar to a given one (see Section 4). This is done using a (dis)similarity function between feature vectors. This, in order to adapt to user preferences and, consequently, to change the behavior of display, cluster, and search tasks, cannot stay fixed regardless of so-far performed user actions. Using a *global*, possibly adaptable, similarity criterion along all the BT is also a poor choice, since it would not be able to contextualize user preferences and to exploit the hierarchical organization of the BT. For these reasons, each node/cluster of the BT uses an adaptable *local* similarity criterion to compare images. In our implementation, we assess the dissimilarity of two histograms p and q as a weighted Euclidean distance, that is:

$$d(p, q; \mathbf{w}_j) = \left(\sum_{i=1}^{32} \mathbf{w}_j[i] (p[i] - q[i])^2 \right)^{1/2} \quad (1)$$

where \mathbf{w}_j is the vector of weights associated to node j (vectors are written in bold throughout the paper).

Clustering algorithms: the initial BT is automatically built applying a clustering algorithm to images descriptors. Currently, we recursively apply the *k-means* algorithm [7] down to the desired granularity level. This, also based on the results in [16], compares favorably to other hierarchical clustering algorithms. Starting with the whole dataset, the *k-means* algorithm is first applied by using the weights, \mathbf{w}_0 , of the root node to compute distances between image descriptors. The so-obtained k clusters are then associated to k children nodes of the tree root, their *local* weight vectors, \mathbf{w}_j , are computed, and the *k-means* algorithm is recursively applied to the clusters whose cardinality is higher than k .

The resulting BT is thus an unbalanced tree whose nodes correspond to clusters of image descriptors. For each cluster C_j , the weight vector \mathbf{w}_j is computed as $\mathbf{w}_j[i] = 1/\sigma_j[i]^2$, where $\sigma_j[i]^2$ is the variance of feature vectors in C_j along the i th coordinate of the N -dim space.

When the user performs a personalization action on the BT (see also Section 5), it moves images from/to clusters associated to nodes of the tree. As a result, the number of children of a node may deviate from its initial value, thus the arity of the BT is not fixed. The initial arity of BT is chosen so as to obtain well-separated clusters and to avoid the cluttering of images in the GUI. To this end, we used the *Davies–Bouldine* cluster validity index [7] and found that a value in the interval [8, 12] is appropriate for the data used in our experiments.

Finally, we would like to stress the fact that, by using weights that are local to each node, the BT is able to “learn” the right distance function that has to be used to compare two image descriptors, according to the content of each cluster, since the most specific available distance is applied. This, combined with the cluster merging/splitting personalization actions, gives PIBE the ability to perform a “local” restructuring of a cluster in a way that is both efficient, because only images in that

cluster are taken into account, and effective, because the most effective distance is used for such images [3].

3 The browsing tree

Nodes of a BT (part of a sample BT is shown in Fig. 3) come in two flavors: *ImgNode*'s are nodes containing (the reference to) a single image, whereas each *ClustNode* is associated to a cluster and contains pointers to children nodes, as well as some concise information, the so-called *clustering summary*, about images in the cluster.

Definition 1 (Clustering Summary) The Clustering Summary, CS_j , of cluster C_j is defined as $CS_j = (M_j, S_j, SS_j)$, where $M_j = |C_j|$, $S_j = \sum_{p \in C_j} p$, and $SS_j[i] = \sum_{p \in C_j} p[i]^2$.

Clustering summaries are extremely helpful, since they provide all the information needed to browse/personalize the BT. In particular, the cluster *centroid*, c_j , is obtained as $c_j = S_j/M_j$, and local weights w_j are computed as $w_j[i] = 1/\sigma_j[i]^2 = 1/(SS_j[i]/M_j - (S_j[i]/M_j)^2)$. For visualization purposes, each cluster C_j has associated an image of the dataset, r_j , called the *representative image* of C_j . This is the image in C_j that is closest to c_j , according to the local distance function.

The structure of BT nodes is described, using an object-oriented pseudo-code, in Fig. 4. It has to be noted that, in order to help the navigation within the BT, each node also contains the pointer to its parent node. Moreover, as Fig. 3 shows, the root

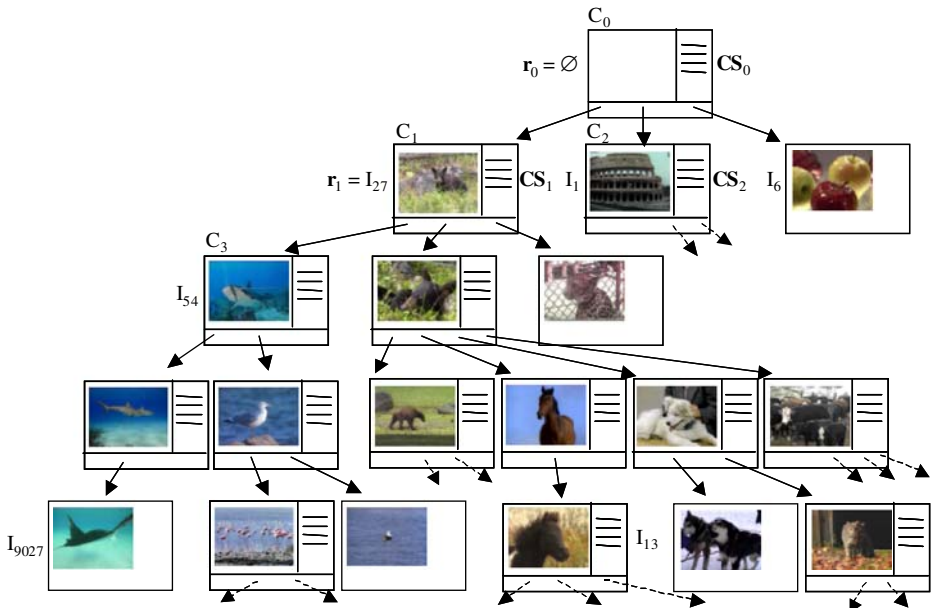


Fig. 3 Part of a sample Browsing Tree

```

class ClustFeature {
  M: int;          // cluster cardinality
  S: double[N];   // sum of feature vectors
  SS: double[N];  // sum of squared feature vectors }
class ImgNode {
  id: int;        // image identifier
  p: double[N];  // feature vector
  parent: ClustNode; // pointer to parent node }
class ClustNode extends ImgNode {
  CF: ClustFeature; // clustering feature
  children: ImgNode[]; // pointers to children nodes }

```

Fig. 4 Structure of BT nodes (N is the dimensionality of image descriptors)

node of the BT does not have a representative image (having an image representing the whole dataset is useless) and its parent node is null.

Personalization actions have the net effect of inserting/deleting images into/from a cluster, thus we need a way to quickly update information local to each node affected by an action. The following theorem demonstrates that the clustering summary CS'_j of a modified cluster C_j can be obtained by simply adding/subtracting feature vectors to/from CS_j , and that the clustering summary of a parent cluster is the sum of the summaries of its children, respectively.

Theorem 1 *When a cluster C_k is added (resp. subtracted) to a cluster C_j , the corresponding clustering summary, CS_j , is modified as follows: $CS'_j = CS_j + CS_k = (M_j + M_k, S_j + S_k, SS_j + SS_k)$ (resp. $CS'_j = CS_j - CS_k = (M_j - M_k, S_j - S_k, SS_j - SS_k)$).*

The clustering summary, CS_p , of a cluster with children C_{j_1}, \dots, C_{j_m} is: $CS_p = \sum_{i=1}^m CS_{j_i} = (\sum_{i=1}^m M_{j_i}, \sum_{i=1}^m S_{j_i}, \sum_{i=1}^m SS_{j_i})$.

4 Browsing

As described in Fig. 1, the *Browsing Engine* is the basic component of the PIBE architecture. It consists of the *Visualizer* and the *Browsing Processor* that together implement the navigation mechanism. The Browsing Processor is also in charge of the management of BT updates.

To show the content of the image DB, the Visualizer adopts a spatial visualization approach, where high dimensional feature vectors are mapped on the 2-D screen (or on the 3-D viewing room), rather than presenting them in a sequential way, to highlight image similarity [11]. In particular, among the available techniques in PIBE we have chosen to adopt *Multidimensional Scaling* (MDS) [12].

To generate the image layout, MDS needs as input all the dissimilarities between the involved images (i.e., representative images for ClustNode objects of the BT, or individual images for ImgNodes). For each set of images to be displayed, a *specific* distance is used. Intuitively, if we have to display images belonging to cluster C_j , a natural choice is to use w_j to compare them. Following this, if images coming from different clusters are to be compared, the distance function of the *most specific cluster* containing all such images should be used.

Definition 2 (Least Common Ancestor) Given a set of images, \mathcal{S} , their Least Common Ancestor (LCA), $C_{LCA(\mathcal{S})}$, is defined as the BT cluster for which $\mathcal{S} \subseteq C_{LCA(\mathcal{S})}$ and $\exists C_j \subset C_{LCA(\mathcal{S})} : \mathcal{S} \subseteq C_j$.

Thus, every time we have to display a set of images \mathcal{S} , $w_{LCA(\mathcal{S})}$ is used in Eq. (1). For the example of Fig. 3, to display images I_{54} , I_{9027} , and I_{13} we use w_1 , whereas for images I_6 , I_{27} , and I_{54} we use w_0 .

PIBE provides two browsing modalities to explore the DB content:

Vertical: The user selects an image on the display (by *clicking on* it with the left mouse button) and *zooms in* the cluster content (i.e., the images representing the children clusters are shown). This modality is the traditional top-down way of browsing a hierarchical structure. PIBE also gives the user the opportunity to view the content of an entire subtree. To do this, the user clicks on a cluster representative image with the right mouse button and provides, as input, a maximum number m of images to be displayed (in order to avoid cluttering the screen if the subtree contains a large number of images). The Browsing Processor will then select the m images in the subtree which are most similar (according to the local distance function) to the cluster representative. Figure 5(a) shows the result of a vertical exploration, where the user has selected the image pointed by the red arrow in Fig. 2.

Horizontal: Vertical browsing only allows a top-down exploration of the DB content: if the user is interested in something that is similar to two or more cluster representative images, she has to visit all the relevant tree branches one at a time. To overcome such limitations, PIBE includes a novel, so-called *horizontal*, browsing modality, that allows the user to also explore the regions of the *empty* space where no representative image is present. When the user clicks with the right mouse button on a point in the empty space, she is required to enter the maximum number m of images to be displayed. The Browsing Processor then retrieves all the images contained in the next

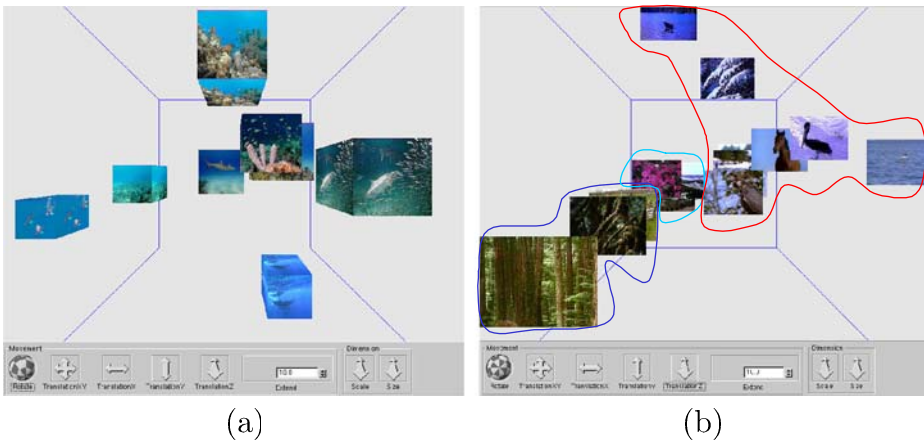


Fig. 5 Results of (a) vertical and (b) horizontal exploration on the example of Fig. 2 after having switched to 3-D display. In (b) images in the same child cluster are highlighted

BT level and, using the weights of the LCA node, returns the m images closest to the selected point (if the next BT level contains $m' < m$ images, only such m' images are shown). In the example of Fig. 2, if the user clicks on the red cross and selects $m = 20$, the children of all displayed clusters are mapped on the displayed space; among those images, only the 20 that have been mapped closest to the selected point are displayed (see Fig. 5(b) for a 3-D display of the result).

Note that if the user executes in sequence h steps of horizontal browsing, the displayed (representative) images are guaranteed to belong to clusters whose LCA is no more than $h + 1$ levels up in the BT. In particular, with just one step we are guaranteed to see images that all have a same grand-parent node.

5 Personalization actions

The browsing facilities of PIBE offer the user powerful ways to explore the BT to look for images of interest. However, the initial BT, which is built in an automatic way, may not reflect the meaning that different users may attach to clusters and individual images. To bridge such semantic gap, PIBE also provides a number of actions that a user can perform to adapt the initial BT to her current preferences. We identified some basic conceptual problems that may arise in a BT: for each of them PIBE provides a specific personalization action to cope with.

- The most common situation is when a node of the BT is not semantically related to its parent. In this case, we want to move the entire node subtree to a different part of the BT (Fusion action).
- Another problem arises when sibling nodes are not semantically related to each other, thus their parent node cannot be considered as an appropriate generalization of them. The Split action allows the user to drop the parent node and substitute it with two or more selected children.
- A final issue arises when a BT node is too specialized, and we want to reduce the number of its children (Merge&Divide action).

Note that, since all personalization actions are *local*, i.e., they are limited to the portion of the BT affected by the user interaction, costly global reorganizations of the BT are avoided.

Although personalization actions could be regarded as independent of each other, it is useful to view them as obtained by combining some basic *primitive operations* acting on the BT. These are:

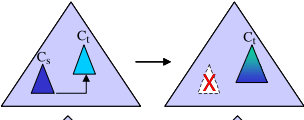
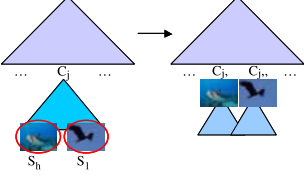
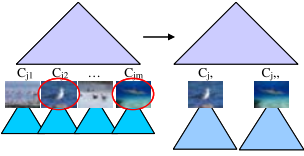
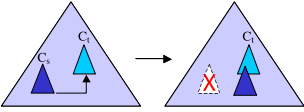
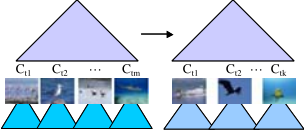
Delete: Deletes an *ImgNode/ClustNode* from its parent node. The clustering summaries *CS* of the ancestors of the deleted node (up to the root) can be immediately updated due to Theorem 1.

Insert: Inserts an *ImgNode/ClustNode* as a child of a *ClustNode* C_j . Again, the *CS*s of C_j and its ancestors can be updated by way of Theorem 1.

Partition: Given a cluster C_j and a set of image descriptors \mathcal{S} , partitions the images in C_j using w_j , assigning each image $I \in C_j$ to its closest image in \mathcal{S} . Its output is a set of $|\mathcal{S}|$ clusters.

Clustering: Given a cluster C_j and an integer k , recursively applies the k -means algorithm to C_j . It outputs a subtree rooted in C_j .

Table 1 Personalization actions in PIBE

Action	Description	Command
<p>Fusion</p> 	<p>C_s is merged with C_t; C_t is then reclustered</p>	<p>drag&drop</p>
<p>Split</p> 	<p>C_j is partitioned by using selected images; resulting subtrees are item clustered and used to replace C_j</p>	<p>select images, “Split” menu</p>
<p>Merge&Divide</p> 	<p>C_j is partitioned by using selected images; resulting subtrees are clustered and used to replace children of C_j</p>	<p>select images, “Cluster” menu item</p>
<p>Graft</p> 	<p>C_s is moved under C_t; no reclustered occurs</p>	<p>drag&drop with Shift key</p>
<p>ReCluster</p> 	<p>C_t is reclustered; the resulting k subtrees replace children of C_t</p>	<p>“Cluster” menu item</p>

The personalization actions can be consequently described in terms of primitive operations as follows (see also Table 1):

Fusion: This action allows the user to merge two clusters of images.² On the PIBE GUI, the user selects the (representative image of the) *source* cluster, C_s , to be moved and drag it over the (representative image of the) *target* cluster, C_t . The Browsing Processor will Delete C_s and Insert it into C_t ; further, the representative image for C_t , r_t , will be chosen using the updated CS_t and a Clustering operation will be applied to C_t using the default k value, thus obtaining a new subtree rooted at C_t . As a net effect, the organization of the target cluster gets updated by also using the new images that have been moved into it.

Split: To split a cluster C_j , the user has first to select through the GUI those images S that should replace C_j , and then to activate the reclustering of C_j by selecting “Split” in the “Reorganize” menu. As a result, cluster C_j will be replaced by $|S|$

²This was the only personalization action included in the first version of PIBE [1] where it was termed “Moving Cluster.”

clusters obtained from the Split. The involved primitive operations are as follows: first, C_j is Deleted, then images in C_j are Partitioned using the selected images, \mathcal{S} , as seeds; finally, Clustering is performed on the $|\mathcal{S}|$ so-obtained clusters and resulting subtrees are Inserted as children of the parent node of C_j .

Merge&Divide: To reduce the number of children of a cluster C_j , the user can choose through the GUI those images \mathcal{S} that should become the new children of C_j , and recluster C_j , by clicking on the “Cluster” item in the “Reorganize” menu. So-obtained subtrees replace the old children of C_j . The detailed primitive operations are: all the children of C_j are Deleted, images in C_j are Partitioned using \mathcal{S} as seeds; the $|\mathcal{S}|$ result clusters are then fed to Clustering and corresponding subtrees Inserted as children of C_j .

Even if the above-described actions allow for an effective personalization of the BT, it might be the case that the user wants to reorganize the BT at a finer-than-cluster granularity level, e.g., by moving single images from a cluster to another. In principle, this fine tuning can be seen as a particular Fusion action, where only the representative image of a cluster C_s, r_s , rather than C_s as a whole, is moved to a target cluster C_t .³ Consider, however, performance aspects, and assume that the user wants to move into cluster C_t , say, ten images that are initially found in different parts of the BT: each of the corresponding ten Fusion actions would then lead to recluster C_t from scratch, thus a lot of repeated work is expected to occur. This observation motivates the introduction of two additional actions that decouple the task of moving images (or even clusters) from that of reorganizing the target cluster. In this way, the user can first build an “ideal” cluster C_t by moving images into it, and then reorganize it once.

Graft: If the user drags an image with the Shift key pressed, this has the only effect to Delete it from its original position in the BT and to Insert it as a child of the target cluster without performing a Clustering. The user can choose whether moving the whole cluster C_s or just its representative image r_s by means of the mouse button used to perform the drag&drop operation (left button=move whole C_s , right button=move just r_s). In the latter case, a new representative image is computed for C_s , by using the updated CS_s .

ReCluster: With ReCluster, the representative image of C_t, r_t , is computed using CS_t and a Clustering is applied to images in C_t .

6 Evaluation

Results in this section were obtained on a dataset of over 8.000 images, extracted from the IMSI collection (IMSI MasterPhotos 50,000: <http://www.imsisoft.com>). We first provide some intuition on the effects that the use of adaptable distance functions

³Indeed, this was the case with the first PIBE version, where this was termed a “Moving Image” action and implemented with the same logic of “Moving Cluster.”

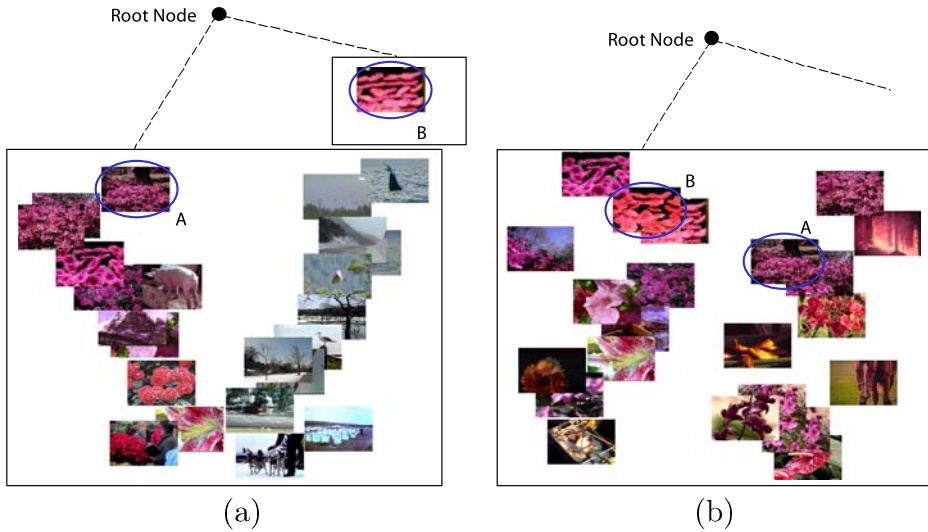


Fig. 6 Display windows after one horizontal browsing step: (a) BT with Euclidean distance; (b) initial BT (using weights)

has on the overall organization of images in the BT. As an example, Fig. 6 shows two snapshots obtained from (a) a BT built using the Euclidean distance and (b) the initial BT that uses weights inversely proportional to the variance of feature values. Consider images A and B, both representing flowers. While in Fig. 6(b) they are put in sibling clusters, thus visible together after an horizontal browsing step, this is definitely not the case in Fig. 6(a), where A and B are quite far apart in the BT. More in general, if the user is looking for flowers, the clusters in Fig. 6(b) attain a precision of 18/23, whereas this drops to 9/23 when weights are not used. This extends to all other image categories in the dataset.

Experiments performed on the first version of PIBE [1] confirmed that, by using the Fusion action alone, users were able to improve the quality of the BT, reducing the time needed for browsing. In the following experiments we evaluate the usability of PIBE, testing it on nine subjects (six males and three females, two non-computer experts, average age of 27.1 years). After a short demonstration of the PIBE functionalities, subjects were asked to conduct two types of experiments:

1. “Find a given image in the BT” (time limit: 5 min);
2. “Find as many images as possible in a given semantic category, e.g., sharks” (3 min).

Experiments were performed on the initial BT (Initial) and on a customized BT (Custom) that was modified by an expert, by way of a number of actions, so as to group together images based on their semantic content. In particular, with just four actions, the expert was able to obtain a cluster containing 54 (out of 70) images of the target category (“fishes in the sea”), considerably improving over the 39 relevant images in the original cluster.

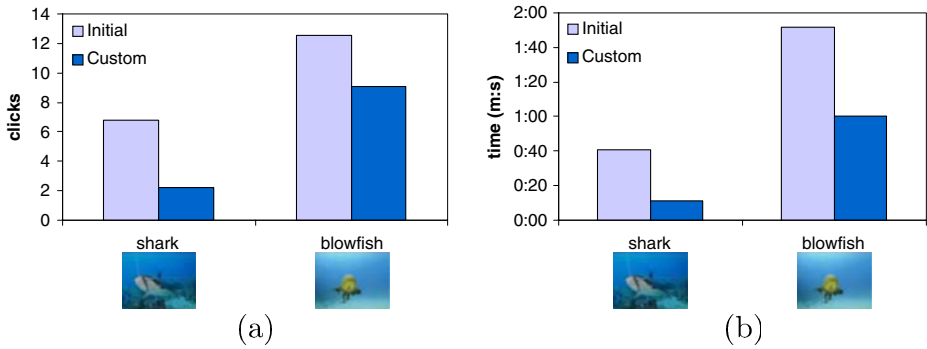


Fig. 7 Average completion (a) work (i.e., mouse clicks) and (b) time for tasks of type one for the “shark” and the “blowfish” images

Figure 7 shows average completion work, evaluated as number of mouse clicks needed to reach the target image, and elapsed time for two tasks of type one. In both cases, all users were successful in finding the requested images within the prescribed time limit, which confirms that the initial BT already represents a good solution. However, results obtained with the Custom BT show that the goal is reached faster, with an average improvement of 72.05 and 45.92%, respectively, and with less work (average improvement of 67.21 and 27.43%, respectively). The difference of results for “shark” and “blowfish” images strongly depends on the corresponding color distribution. Due to the important yellow component, it is harder to classify “blowfish” as a “fish in the sea” than it is with the “shark”, where blue is the dominant color.

Experiment two was designed to evaluate the performance of PIBE when looking for a set of semantically-related images. Results indicate that the Customized BT was helpful also for this task, with an improvement of 16.13% over the Initial BT (with an average of 61.6 vs. 51.7 “fishes in the sea” images found in 3 min). This, again, demonstrates the usefulness of PIBE personalization actions (remember that the Custom tree was obtained from the initial BT with only four actions).

Finally, with the knowledge on the content of the dataset gained from previous experiments, participants were invited to build their own BT to directly test PIBE personalization actions, and to fill a questionnaire. Table 2 shows that there is an

Table 2 Mean satisfaction scores across participants, using 0 as “strongly disagree” and 4 as “strongly agree”

Statement	Agreement	Average score
	0 1 2 3 4	
a) “I like the browsing system”	0 0 0 3 6	3.67
b) “The system is easy to use”	0 0 2 4 3	3.11
c) “It was easier to find images I wanted in the Custom case”	0 0 1 2 6	3.56
d) “The initial BT needed to be updated to meet my expectations”	0 0 1 7 1	3.00
e) “I found the personalization actions intuitive”	0 0 0 6 3	3.33
Total average		3.33

overall positive agreement from all participants (3.33/4 on average) on all the questionnaire statements (most significant results are in boldface). In particular, (see statement (a) of Table 2) all subjects liked PIBE and (statement (c)) agreed that the Customized tree makes easier to find images of interest. Finally, (statement (e)) participants found the personalization actions very intuitive. By analyzing individual answers, we note that 2 subjects gave score 2 to PIBE usability (statement (b)). Their comments were: “It was not difficult to use but I need a longer practice”, and “At the beginning I found the GUI very sensitive to mouse actions”. Almost all subjects agreed on the need to update the initial BT (statement (d)); only a subject scored 2 and commented: “I agree on the usefulness of personalization actions, yet I was surprised by good results obtainable even with the initial BT”.

Results of experiments are extremely encouraging. The tasks were intentionally hard, because we included, among the participants, subjects that were non-computer experts and/or came from different scientific areas. Most importantly, all participants were unfamiliar with the image collection: we argue that in case the user knows the content of the image DB, as in the case of *personal digital libraries*, results would be even better.

Cost analysis: Finally, we provide some insights about the computational complexity of operations supported by PIBE. A first basic observation concerns the complexity of Clustering. Since the time complexity of (flat) k -means is $O(kn)$ for (re-)clustering n objects, it is easily derived that the recursive k -means has a cost of $O(kn \log_k n)$. This increases monotonically with k , which suggests that limiting the value of k to be used, besides being advisable for avoiding clutter of images on the display, is also beneficial from an efficiency point of view. For instance, using $k = 11$, it requires 9.43 s to build the initial BT over 2,210 images, and 52.59 s for a dataset of 8,272 images.

As discussed in Section 5, Clustering is used by all the basic personalization actions. For the Fusion action the Browsing Processor needs also to update representative images and clustering summaries. Assume that at a given time the GUI displays a set of images, including the representative images of C_s and C_t , and let C_{LCA} be the least common ancestor of C_s and C_t . If a Fusion action takes place at this stage, BT updates remain *local* to C_{LCA} , thus the clustering summary CS_{LCA} will not change at all. The same is clearly also true for the representative image of C_{LCA} . On the other hand, the statistics of each *intermediate* cluster C_j on the paths from C_{LCA} to C_s and to C_t need to be properly updated.⁴ This, thanks to Theorem 1, is immediately obtained by proceeding in a bottom-up fashion. Concerning the computation of the new representative images for the intermediate clusters, this requires to go through all the images in such clusters. This search, which in principle needs to be repeated from scratch for each cluster, can be again optimized using a bottom-up computation in which each intermediate cluster C_j is searched with the centroids of C_j itself and of all its intermediate ancestors.

⁴Note that ordinary vertical browsing guarantees that C_{LCA} is the parent cluster of C_s and C_t , thus no intermediate cluster is present.

7 Related work

The PIBE design integrates, and develops from, basic ideas and principles rooted in the fields of browsing systems and adaptable content-based similarity search. Like other browsing systems [4, 8, 10, 13], it exploits automatically extracted low-level features, rather than user-supplied and/or metadata information [6, 9, 15], which might not be always available, to build a hierarchical organization. This contrasts with a flat view, see e.g., [10], which hardly scales to large databases. The BT also allows the (2-D and 3-D) MDS-based spatial arrangement of images, which is also used in [12], to be efficiently implemented.

Most browsing systems are based on a static structure, in that they do not allow users to modify the default organization of images. PIBE overcomes this limitation by exploiting principles of query refinement, well-established for content-based similarity search [2, 3], which allow the similarity criterion to vary depending on clusters' content and, consequently, on user actions. Remarkable examples of systems that exploit the same idea are El Niño [13], PicSOM [8], and the similarity pyramid [4], which we want to discuss in some more detail.

The main customization tool that the user can use while interacting with El Niño interface is *configuration feedback*. This, given a set of randomly selected images from which the browsing session has to start, allows the user to select images of interest and place them on the screen so as to reflect their (intended) mutual similarities. Then, the system will adapt its (internal) similarity criterion to match the user one and to present her with an updated display of images. Being based only on the current user's goal, El Niño cannot guarantee a stable organization of images based on user preferences. Further, updating the similarity criterion involves the whole DB, which is computationally expensive. Finally, no browsing session can take place if the user does not see any image of interest in the initial random configuration.

PicSOM uses a tree-structured self-organizing map (TS-SOM) to organize images. Low-level image features are used to train the levels of the TS-SOM using a vector quantization algorithm. Eventually, each TS-SOM unit will correspond to a vector which is the average of all images mapped under that unit. Unlike PIBE, PicSOM uses a single similarity criterion to cluster images, which, as arguable from our experiments, is definitely not a good choice. The same remains true if the user wants to search into the DB. In this case, the user is driven towards the most promising parts of the tree by the color of TS-SOM units, which depends on the correlation between images in that part of the tree and the positive/negative image samples selected by the user for her task. This color metaphor does not give the user any clue about the actual images included in the sub-trees and, as said, has no influence on the overall organization of the TS-SOM.

Similarly to PIBE, the similarity pyramid represents each cluster with an icon and lays out images on a 2-D grid based on their mutual similarities. These are modified when the user selects a set, R_i , of images as relevant to her current search task. Given a base distance function, the *worm hole distance* between an image x and R_i is the minimum of base distances between x and images in R_i . Although results in [4] show the potentialities of the approach, this is limited by the number of different search tasks supportable, since each of them defines an equivalence class of images, R_i . The alternative of “forgetting” the relevant sets of past browsing sessions, although

trivially solving the problem, has the drawback to give up a long-term organization of images.

8 Conclusions

In this paper we have presented the PIBE image browsing system, which provides the user with a novel set of browsing and personalization facilities allowing to customize the hierarchical organization of images (the Browsing Tree) in an effective and efficient way. A key feature of PIBE is that it maintains local similarity criteria for each portion of the Browsing Tree. This makes it possible both to avoid costly global reorganization upon execution of user actions and, combined with a persistent storage of the Browsing Tree, to efficiently support multiple browsing tasks.

Acknowledgement This work was supported by the IST-2001-33058 PANDA Project (2001–2004).

References

1. Bartolini I, Ciaccia P, Patella M (2004) The PIBE personalizable image browsing engine. In: Proc. of the 1st international workshop on computer vision meets databases (CVDB 2004), Paris, France, pp. 43–50
2. Bartolini I, Ciaccia P, Waas F (2001) FeedbackBypass: a new approach to interactive similarity query processing. In: Proc. of the 27th international conference on very large data bases (VLDB 2001), Rome, Italy, pp. 201–210
3. Chakrabarti K, Ortega M, Mehrotra S, Porkaew K (2004) Evaluating refined queries in top-k retrieval systems. *IEEE Trans Knowl Data Eng* 16(2):256–270
4. Chen J, Bouman C, Dalton J (1999) Active browsing using similarity pyramids. In: Proc. of international conference on storage and retrieval for image and video databases (SPIE 1999), San Jose, California, pp. 144–154
5. Combs TTA, Bederson BB (1999) Does zooming improve image browsing?. In: Proc. of the fourth ACM conference on digital libraries, Berkeley, California, pp. 130–137
6. Graham A, Garcia-Molina H, Paepcke A, Winograd T (2002) Time as essence for photo browsing through personal digital libraries. In: Proc. of ACM/IEEE joint conference on digital libraries (JCDL 2002), Portland, Oregon, USA, pp. 326–335
7. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice Hall, New Jersey
8. Laaksonen J, Koskela M, Laakso S, Oja E (2000) Self-organising maps as a relevance feedback technique in content-based image retrieval. *Pattern Anal Appl* 2(4):140–152
9. Loui AC, Wood MD (1999) A software system for automatic albuming of consumer pictures. In: Proc. of the 7th ACM international conference on multimedia '99, Orlando, Florida, USA, pp. 159–162
10. Platt JC, Czerwinski M, Field BA (2003) PhotoTOC: automatic clustering for browsing personal photographs. In: Proc. of the 4th IEEE pacific rim conference on multimedia vol 1, Singapore, pp. 6–10
11. Rodden K, Basalaj W, Sinclair D, Wood K (2001) Does organisation by similarity assist image browsing?. In: Proc. of the SIG-CHI human factors in computing systems (CHI 2001), Seattle, Washington, USA, pp. 190–197
12. Rubner Y, Tomasi C, Guibas LJ (1998) A metric for distributions with applications to image databases. In: Proc. of the 6th international conference on computer vision (ICCV 1998), Mumbai, India, pp. 59–66
13. Santini S, Jain R (2000) Integrated browsing and querying for image databases. *IEEE Multimed* 7(3):26–39
14. Smeulders AWM, Worring M, Santini S, Gupta A, Jain R (2000) Content-based image retrieval at the end of the early years. *IEEE Trans Pattern Anal Mach Intell* 22(12):1349–1380
15. Wallace M, Karpouzis K, Stamou G, Moschovitis G, Kollias S, Schizas C (2003) The electronic road: personalized content browsing. *IEEE Multimed* 10(3):49–59
16. Wojna A (2003) Center-based indexing in vector and metric spaces. *Fundam Inform* 56(3):1–26



Ilaria Bartolini is currently an Assistant Professor with the DEIS department of the University of Bologna (Italy). She graduated in Computer Science (1997) and received a Ph.D. in Electronic and Computer Engineering (2002) from the University of Bologna. In 1998 she spent six months at CWI (Centrum voor Wiskunde en Informatica) in Amsterdam (The Netherlands) as a junior researcher, and in 2004 she was a visiting researcher at NJIT (New Jersey Institute of Technology) in Newark, NJ, USA. Her current research mainly focuses on image retrieval/browsing, learning of user preferences, and processing of similarity and preference-based queries. Ilaria Bartolini co-authored many papers published in international journals and conferences (like TPAMI and VLDB). She served in the program committee of several international conferences and workshops and is a member of ACM SIGMOD.



Paolo Ciaccia has a “Laurea” degree in Electronic Engineering (1985) and a PhD in Electronic and Computer Engineering (1992), both from the University of Bologna, Italy, where he has been Full Professor of Information Systems since 2000. He has published more than 80 refereed papers in major international journals (including IEEE TKDE, IEEE TSE, ACM TODS, ACM TOIS, IS, DKE, and Biological Cybernetics) and conferences (including VLDB, ACM-PODS, EDBT, and ICDE). He was PC chair of the 2002 edition of the SEBD italian conference on advanced data bases. His current research interests include similarity and preference-based query processing, image retrieval, P2P systems, and data streams. He was one of the designers of the M-tree, an index for metric data used by many multimedia and data mining research groups in the world. He is a member of IEEE.



Marco Patella got the “Laurea” degree in Electronic Engineering from the University of Bologna, Italy. He received a PhD in Electronic and Computer Engineering (1999) from the same University. Since 2001 he has been a Researcher at University of Bologna with DEIS. His current research interests include similarity-based query processing in multimedia databases and Data Mining techniques. He is one of the designers of the M-tree, an index for metric data, which is used by several multimedia and data mining research groups in the world. He has published more than 20 papers, in the area of database systems, in major international journals (including IEEE TPAMI and ACM TODS) and international conferences (including VLDB, ACM-PODS, EDBT, and ICDE). He has also been part of the program committee of several international conferences and workshops.