20

HIERARCHICAL SPACE MODEL FOR MULTIMEDIA DATA RETRIEVAL

Makoto Onizuka and Shuichi Nishioka
NTT Cyber Space Laboratories
{onizuka, nishioka}@dq.isl.ntt.co.jp

Abstract Database management systems for multimedia data retrieval are becoming more important, as digital videos and cameras increase in popularity. An important feature of multimedia data retrieval is that users rarely specify their first queries exactly, and must clarify what they want by browsing the query results, refining their query by trial and error. It is therefore desirable for a multimedia database management system (DBMS) to develop a rough query result quickly and refine it over time. This paper describes a hierarchical space model for the multimedia data retrieval that is similar to that of the human memory hierarchy. The aim of the hierarchical space model is to improve the similarity retrieval's performance with little loss in query result quality. We implemented the hierarchical space model on the ORDBMS LiteObject and applied it to an image retrieval application. The results of this test proved the efficiency of our hierarchical space model.

1. INTRODUCTION

Database management systems for multimedia retrieval are becoming more important because digital cameras and videos are increasing in popularity and because huge amounts of music and movie data (combined in term multimedia data) are now being stored in electrical format. An important feature of multimedia data retrieval (Gupta and Jain, 1997) is that users rarely have a clear idea of what they wish to retrieve and must clarify their queries by browsing the query results and refining the query condition by trial and error.

For example, suppose that you want to search for clip-art images containing a man operating a computer. You will get many clip-art images in the query result by using a multimedia DBMS. However, were you to review all of these images, you would probably find that only a

H. Arisawa et al. (eds.), *Advances in Visual Information Management*

few of them would fit what you had in mind. There are several possible reasons for this result.

1 You don't have a clear idea of what you want and the above condition "image containing a man who operates a computer" is not exact enough. For example, what you really want may be an image containing a man with mobile computer or a man at work in a computer room. However, you don't realize this until you browsed the images.

2 Even if you have a clear idea of what you want, it is difficult to construct a search condition embodying it. For example, suppose you want an image containing a man with a mobile computer running down Wall Street in New York on a rainy day. It would be difficult to translate such a query to a proper search condition that fits the search engine's format, or to construct a search condition with the proper key images.

3 Even if you have a clear idea of what your search condition should be and can construct a search condition embodying your idea, you probably won't get a correct result because the multimedia DBMS technology has not matured enough.

This paper focuses on the first type of data retrieval problem. One way to solve it is to have the multimedia DBMS develop a query result quickly with little loss of a query result quality, have the user browse the result, and submit a refined query (we refer the first query as *rough retrieval* and refer the conventional method as *precise retrieval*). There are three approaches achieving this rough retrieval.

1 A *hierarchical space* stores instances in a hierarchy that is a metaphor of the human memory hierarchy. Basically, this approach places important instances in upper layers and less important instances in lower layers. Thus, rough retrieval can be achieved by limiting the target instances that are managed only in the upper layers.

2 A *hierarchical data structure* stores an instance at several different resolution levels. FlashPix format and thumbnail data are examples of different resolution levels. Thus, rough retrieval can be done by executing a query using the rough resolution level data. Another way to do rough retrieval is by storing feature vector data extracted from each instance at several different levels. Thus, rough retrieval can be done by executing a query that uses rough feature vector data.

3 Rough retrieval can be done by tuning the parameters of a a multidimensional index algorithm like R-tree (Guttman, 1984).

This paper describes a hierarchical space model based on the *hierarchical space* approach. The main feature of the space model is the introduction of a layer concept to divide the instances into several collections of instances for each class, whereas conventional models manage all instances in a single space. The search space of the hierarchical space model is reduced by specifying the upper layer and keeps a query result quality by using appropriate division rules.

2. HIERARCHICAL SPACE MODEL

2.1. DATA MODEL

Class. Class is a template of an *instance* (*class* instantiates an *instance*) and is composed of name, ID, the maximum depth of the hierarchy (depth), a collection of *attribute*s (attributes), and a collection of *layer*s (layers).

Each *attribute* is composed of name and type and *method* is composed of name and return type.

$class = \langle$ name *string*, ID *integer*, depth *integer*, attributes *Set*(*attribute*), layers *Set*(*layer*)$\rangle$
$attribute = \langle$ name *string*, type *type*$\rangle$

where *Set*(v) is a value collection of type v.

Layer. Layer manages a collection of *instance*s (instances), the depth of itself in the hierarchy (number), the reference to the class that the instances are instantiated from, and layer type.

$layer = \langle$number *integer*, instances *Set*(*instance*), class *class*, type *ltype*$\rangle$

There are two types of layers: 1) real layers for persistent instances, and 2) virtual layers that are created by performing layer operations on existing layers. A number is given to each layer that indicates its depth in the hierarchy.

There are two types of useful hierarchies: 1) **recall/forget hierarchy** represents the level of importance of an instance. The less important an instance is, the lower it goes in the hierarchy. Figure 1 shows an example of this type of hierarchy (storing *database research* class instances) in which *multimedia* is the most important instances and *OLTP* and *RDB* are the least important instances in the database. 2) **abstract/concrete hierarchy** represents the level of abstraction of

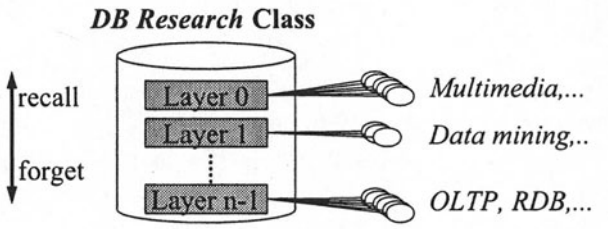

Figure 1 Recall/forget level hierarchy

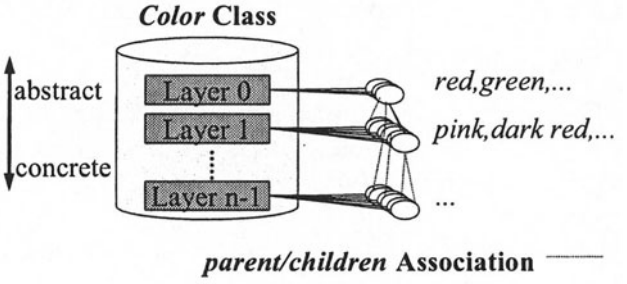

Figure 2 Abstract/concrete level hierarchy

an instance. The more abstract an instance is, the higher it is placed in the hierarchy. Figure 2 shows an example of this type of hierarchy (storing *color* class data) in which *red* and *green* are in the highest layer of abstraction and *pink* and *dark red* are in lower layers of abstraction. The parent/children association between instances is a self-association in this case.

The *association* and *association instance* are described below to support the abstract/concrete hierarchy.

Association. Association is a directional reference between two classes and is a template of an *association instance* (*association* instantiates *association instance*). It is also composed of name, ID, the class reference of its source class (from), the class reference of its destination class (to), and the association type. The association type can be one to one, one to many, many to one, or many to many. The operation called traverse (represented as *trav*) navigates from a source instance to destination instances through the association.

association = ⟨ name *string*, ID *integer*, from *class*,
to *class*, astype *astype*⟩

$trav : association \times instance \rightarrow Set(instance)$

2.2. OPERATIONS

Union. The *union* operation generates a virtual layer from two layers. The generated layer manages the instances that form the union of the layers' instances.

$$layer(n, inses_n) \text{ union } layer(m, inses_m) = layer(l, inses_n \ union \ inses_m)$$

Difference. The $difference$ (represented as $-$) operation generates a virtual layer from two layers. The generated layer manages the instances of the first layer that are left over after the instances that are common to both layers are removed.

$$layer(n, inses_n) \text{ - } layer(m, inses_m) = layer(l, inses_n - inses_m)$$

Selection. The *selection* (represented as σ) operation generates a virtual layer from another layer. The generated layer manages the instances that satisfy a specified predicate.

$$\sigma_p(layer(n, inses_n)) = layer(l, inses_l)$$

where $inses_l \subset inses_n$ and $\forall ins \in inses_l . p(ins) = true$.

Redistribute. The redistribute (represented as *redist*) operation redistributes instances from one specified layer to another specified real layer.

redist(targetLayer *layer*, toLayer *layer*)

This operation deletes all instances of the targetLayer in all real layers except the toLayer and moves all instances of the targetLayer to the toLayer.

3. IMPLEMENTATION

We implemented the hierarchical space model on the ORDBMS LiteObject. This section describes the details of the implementation including the redistribution process in two types of hierarchies and gives results of experiment conducted on the implementation.

3.1. REDISTRIBUTION IN HIERARCHIES

Recall/forget hierarchy. We implemented this hierarchy by storing the number of times each instance is accessed and by using an embedded function *getAccessCount()* to get this number. For example, in

Figure 1, suppose that the *DB research* class has two layers and that you want to redistribute instances using the rules given below. If the number of times an instance is accessed is larger than a specified threshold (ten in the example), the instance is moved to layer 0, and rest of the instances are moved to layer 1. This function is represented by the SQL statements below.

> **redist class** *DB Research* **layer** 0,1 **where** *getAccessCount()* $\leq$ 10 **to layer** 0; **redist class** *DB Research* **layer** 0,1 **where** *getAccessCount()* $>$ 10 **to layer** 1;

Abstract/concrete hierarchy. An example of this type of hierarchy is shown in Figure 2. Suppose that the *Color* class in the figure has several layers and that you want to redistribute instances according to their depth along the path of the a *parent/children* association. In this case, instances that don't have parents are moved to layer 0, and instances that are traversable via the *parent/children* association from instances in layer 0 are moved to layer 1, and so forth. Those instances whose path lengths exceed the maximum depth of the *Color* class hierarchy are moved to the bottom layer. This function is represented by the SQL statement below.

> **redist class** *Color* **via** *parent/children*;

3.2. IMAGE RETRIEVAL APPLICATION

Retrieval process. First, we construct an initial query that has a rough condition because, as explained in section 1, we still don't have a clear idea of what we want at this point. Second, we do a rough retrieval using the initial query condition and get the result. Third, we refine the query condition by browsing the result and then execute the refined query. We repeat this step until we get the desired result.

Rough retrieval is useful until the query is clarified, because it returns an answer faster than conventional precise retrieval.

Image retrieval application. Figure 3 shows the image retrieval process of the ExSight, the left side of the figure shows the GUI of the ExSight viewer, and the right side shows the LiteObject with an access method for still images. The retrieval process is as follows.

1 The user inputs key images into the search condition area of the ExSight viewer and specifies the target layers that represent the rough level of the query. For example, the search condition area in Figure 1 shows a situation in which the user is searching for images containing both a boy's face and a blue shirt and has specified layers 0 and 1 (of a three layer hierarchy).

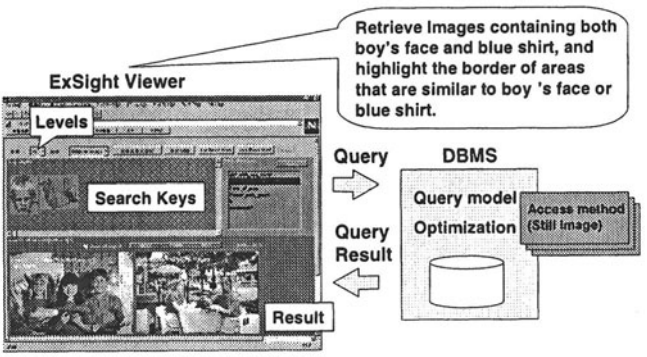

Figure 3 ExSight image retrieval process

2 The ExSight viewer submits a query statement to the LiteObject. To speed up processing, sub-image instances and their feature vector are extracted from each image instance, and multidimensional indices are constructed from the feature vectors (hue, saturation, intensity, outer-shape, size, position) (Curtis et al., 1997).

3 The Liteobject processes the query statement using the specified layers and the multidimensional indices and returns the query result.

4 The ExSight viewer displays the query result. For example, the query result area in Figure 1 shows the top two image instances and highlights the border of the sub-image instances that satisfy the query condition.

3.3. EXPERIMENTS

We compared the performance of the hierarchical space model with that of the conventional model.

Environment. Sun Ultra60 (Ultra SPARC-II 360 MHz * 2), O.S. Solaris2.6, Memory 2 GB.

Data. We used 996 PhotoDisc(R) images (scenery, food, people playing sports and so forth) and 96061 sub-images that were extracted from these images.

Queries. We chose 18 queries that returned good results in the last step of the retrieval process in section 3.2. The queries were categorized

Table 1 Experimental Result

model	image	sub-image	response time(sec)
conventional	996	96061	37.3
hierarchical	{619, 377}	{1651, 94410}	2.2

according to two patterns: 1) images searched using a condition with image keys (three queries) and 2) images and sub-images searched using a condition with sub-image keys (fifteen queries).

Evaluation method. We evaluated the two models as follows.

Conventional: We measured the total response time of all 18 SQL queries after building multidimensional indices.

Hierarchical: We distributed instances according to the number of accesses each of the 18 SQL queries had for each instance, and then measured the total response time of all 18 SQL queries after building multidimensional indices and specifying the upper layer.

Experimental result. Table 1 shows the the result of the redistribution. The image hierarchy comprised 377 instances in the upper layer and 619 instances in the lower layer, and the sub-image hierarchy comprised 1651 instances in the upper layer and 94410 instances in the lower layer.

Our hierarchical model was about 17 times faster than the conventional model because the search space of the hierarchical model was reduced by specifying the upper layer. In addition, there was little difference between the query results of the two models, which demonstrates that the hierarchical model didn't degrade the query result quality. [1]

4. DISCUSSION

Although the implemented division rule is rather simple (based on the number of times each instance is accessed), the experimental result demonstrated the efficiency of our hierarchical model for the image retrieval application ExSight. There are two reasons for this result.

[1] What difference there was was caused by changes in the target instances, which altered the multidimensional indices' structure.

- There are instances that are useless or that have never been searched and those instances are redistributed into the lower layers of the hierarchy. For example, some of the sub-images that the ExSight extracts are useless (e.g., a very small extracted sub-image is almost always useless).

- The number of times an instance appears in the query results (instance frequency) varies from instance to instance. For example, if users tend to search newer data more often than older data, it is possible to improve search performance by redistributing the newer data to the upper layers and the older data to the lower layers.

On the other hand, our model does not improve performance when the instance frequency does not vary much from instance to instance because there is no effective layer division in this case.

There are several problems that remain to be solved.

1. Humans tend to remember what is very important or impressive to them. The implementation of the recall/forget hierarchy does not rank instances according to such a subjective measure; it only ranks an instance according to the number of times it was accessed or by the last time the it was accessed. Relevance feedback or other techniques will solve this problem.

2. The implementation of the abstract/concrete hierarchy redistributes instances according to their depth along the path of the a specific association. It is useful but it is not easy to move each instance to the desired layer. In this case, we have to add an extra attribute to each instance and give a specific redistribution rule even though it is time consuming work.

3. The hierarchical space model can not be adjusted to fit any user profile or context because it handles only one hierarchy at a time. Extending the hierarchical model so that it can support several hierarchies at once will enable effective user-dependent or context-dependent rough retrieval.

4. It is not easy for users to specify which layers in an SQL statement would be best for conducting the rough retrieval. We should build a translator that takes as input the desired response time or the preciseness of the retrieval and calculate the optimum number of layers for the query.

5. CONCLUSION

This paper described a hierarchical space model for multimedia data retrieval that is suitable for users who generally clarify their queries by browsing the query results and refining the query conditions. The main feature of the model is the introduction of a layer concept to divide the instances into collections of instances for each class in a manner similar to that of the human memory hierarchy. We defined the data model and operations based on this layer concept and described two typical hierarchies: recall/forget hierarchy and abstract/concrete hierarchy. We implemented the hierarchical model on the ORDBMS LiteObject using the extended query language and applied it to ExSight for PhotoDisc(R) image data. Our space model improved the similarity retrieval's performance with little loss in retrieval quality.

References

Curtis, K., Taniguchi, N., Nakagawa, J., and Yamamuro, M. (1997). A comprehensive image similarity retrieval system that utilizes multiple feature vectors in high dimensional space. In *Proc. Int. Conf. on ICICS*, pages 180–184.

Gupta, A. and Jain, R. (1997). Visual information retrieval. *Communications of the ACM*, 40(5):71–79.

Guttman, A. (1984). *R*-trees: a dynamic index structure for spatial searching. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 14(2):47–57.

Biographies

Makoto Onizuka Makoto Onizuka received his B.S. degree in Computer Science from the Tokyo Institute of Technology in 1991. He joined NTT Information Systems Laboratories, in 1991, where he was involved in research on relational database design from 1991 to 1992. Since then, he had been working on object-oriented analysis and design, object-oriented database design, and database visualization. He is currently involved in research on an object relational database management system at NTT Cyber Space Laboratories. He is a member of IPSJ and ACM.

Shuichi Nishioka Shuichi Nishioka Received his B.E. degree from Yokohama National University in Japan, in 1995. He then joined NTT Information and Communication Systems Laboratories where he has been engaged in research on an object relational database management systems. Mr. Nishioka is a member of IPSJ.