Proceedings of

mis'01 Capri 7° WORKSHOP

MULTIMEDIA INFORMATION SYSTEMS

7 - 9 November 2001
Villa Orlandi - Capri - Italy

Edited by:
S. Adali, S. Tripathi

AD NUMBER	DATE	DTIC ACCESSION N

1. REPORT IDENTIFYING INFORMATION

A. ORIGINATING AGENCY
Univ. Degli Studi de Napoli Federico II, ITALY

B. REPORT TITLE AND/OR NUMBER
Multimedia Information Systems

C. MONITOR REPORT NUMBER
N68171-01-M-6165

D. PREPARED UNDER CONTRACT NUMBER
RoD 9206-EE-03

2. DISTRIBUTION STATEMENT

APPROVED FOR PUBLIC DISTRIBUTION

DISTRIBUTION UNLIMITED

PROCEEDINGS

REQUES

1. Put your r on revers
2. Complete
3. Attach for mailed tc
4. Use uncl informat
5. Do not or for 6 to

DTIC:

1. Assign
2. Return

20020215 126

DTI OCT 95

DITIONS ARE OBSOLETE

Proceedings of

mis '01 7° WORKSHOP

MULTIMEDIA INFORMATION SYSTEMS

7 - 9 November 2001
Villa Orlandi - Capri - Italy

Edited by:
S. Adali, S. Tripathi

R&D 9206-EE-03
N68171-01-M-6165
Universita Degli Studi di Napoli Federico II, Italy
Proceedings

Under the auspices of:
Regione Campania
Provincia di Napoli
Comune di Napoli
Apt di Capri

With the contribution of:

Università degli Studi di Napoli "Federico II"
Polo delle Scienze e delle Tecnologie
Dipartimento di Informatica e Sistemistica

Università di Salerno
Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica

Computer Associates™

ORACLE

ANDERSEN

Liguori Editore

Preface

We welcome you to MIS 2001, the 7th workshop on Multimedia Information Systems.

MIS 2001 is the seventh of a series of workshops that started in 1995 with the aim of fostering interdisciplinary discussions and research in all aspects of multimedia information systems, in all their diversity. MIS 2001 will be held in Capri (Italy), November 7-9, 2001.

MIS 2001 follows upon the success of the six workshops in this series that were held in Arlington (VA), West Point (NY), Como (Italy), Istanbul (Turkey), Indian Wells (CA) and Chicago(IL).

The aim of this workshop is to bring together experts in all aspects of multimedia information systems, digital media content, multimedia database systems, networking, real-time systems, graphics and visualization, artificial intelligence, and algorithms.

For this workshop we received about 50 papers submitted from about 10 countries around the world. More then 20 reviewers were proposed by the members of the program committee. Each paper was sent to three reviewers whose expertise matched the topic of the papers. It was decided to accept 20 high quality research papers.

During MIS2001 we will have two invited talks by V.S. Subrahmanian and Maria-Luisa Sapino, 6 research sessions on Image Retrieval, Query Processing and Indexing, Multimedia Presentations, Multimedia Information Retrieval, Multimedia Networking and Streaming, Data Models, and a panel session about the new trends in Multimedia Computing. We hope you will find the technical program interesting and stimulating.

We would like to thank the Steering Committee, V. S. Subrahmanian, S. Tripathi and D. Hislop for their leadership in organizing this workshop. We would also like to express our deep gratitude to all the organizations which have given their financial support to this conference, namely the Army Research Laboratories, European Army Research Offices, University of Naples "Federico II", University of Salerno - Department of Information and Electric Engineering, IBM Italia, IntelTec, Computer Associates, and Liguori Editore.

We hope you will enjoy your visit to wonderful Capri, and we hope to meet you again for MIS 2002.

Rama Chellappa	*Sibel Adali*	*Angelo Chianese*
Lucio Sansone	*Satish Tripathi*	*Antonio Picariello*
Workshop general chairs	**Program Chairs**	**Organizing Committee**

A Robust Technique to Recognize Objects in Images, and the DB Problems it Raises

Laurent Amsaleg
IRISA-CNRS
Laurent.Amsaleg@irisa.fr

Patrick Gros
IRISA-CNRS
Patrick.Gros@irisa.fr

Sid-Ahmed Berrani*
IRISA-TMM
Sid-Ahmed.Berrani@irisa.fr

Abstract

Traditional content-based image retrieval systems typically compute a single descriptor per image based for example on color histograms. The result of a query is in general the images from the database whose descriptors are the closest to the descriptor of the query image. Systems built this way are able to return images that are globally similar to the query image, but can not return images that contain *some of the objects* that are in the query. As opposed to this coarse-grain recognition scheme, recent advances in image processing make fine-grain image recognition possible, notably by computing local descriptors that can detect similar objects in different images. Obviously powerful, fine-grain recognition in images also changes the retrieval process: instead of submitting a single query to retrieve similar images, multiple queries must be submitted and their partial results post-processed before delivering the answer. This paper first presents a family of local descriptors that support fine-grain image recognition. These descriptors enforce robust recognition, despite image rotations and translations, illumination variations, and partial occlusions. Many multi-dimensional indexes have been proposed to speed-up the retrieval process. These indexes, however, have been mostly designed for and evaluated against image databases where each image is described by a single descriptor. While this paper *does not present any new indexing scheme*, it shows that the three most efficient indexing techniques known today are still too slow to be used in practice with local descriptors because of the changes in the retrieval process.

1 Introduction

Image processing and database (DB) techniques are together required to build large content-based retrieval systems. Image processing techniques are needed to extract *descriptors* encoding information found in images. Descriptors are typically vectors of real numbers defining points in a high-dimensional space. The similarity of two images is assumed to be proportional to the similarity of their descriptors, which is measured as the distance between the points defined by the descriptors. Similarity search is therefore implemented as a nearest-neighbor search or as a ε-range search within the feature space.

Traditional methods for computing descriptors include color histograms and correlograms [19, 13]. These schemes, used in QBIC [7], Virage, or Excalibur, typically compute a *single* descriptor per image. One descriptor therefore encodes information that is global to one image. In this context, content-based retrieval is performed at a coarse-grain level: the system returns the images that are *globally* similar to the query image. The system can not detect, however, that two images contain similar objects, but at different locations, in front of different backgrounds, from different viewpoints or differently illuminated. To address this problem, modern image processing techniques have recently focused on fine-grain image recognition. Fine-grain –or object– recognition in images typically requires to compute many descriptors per image. These descriptors are often called *local descriptors* because one descriptor encodes information that is local to a (small) area of an image.

The first contribution of this paper is the description of a method designed for fine-grain image recognition that is very robust to changes in color images. This method computes local descriptors that are well suited for detecting similar objects in images despite orientation changes (rotations), translations, illumination variations and partial occlusions.

In general, fine-grain image recognition impacts the retrieval chain when searching for similar images. First, it increases the size of the DB: Instead of storing one descriptor per image, many of them must be stored for each image. The size of the DB is typically increased by two order of magnitude. Second,

*Sid-Ahmed Berrani is supported by a Fellowship from Thomson Multimédia R&D France, 1 avenue Belle Fontaine, BP 19, 35511, Cesson-Sévigné.

it changes the way similar images are searched: Instead of searching the descriptors that are close to the unique query descriptor (as it is the case with global descriptors), the system starts by computing all the descriptors that describe the query image and then queries the DB many times, each time using a different local query descriptor. Each (partial) answer is kept around and once all query descriptors have been used, answers are cross-checked and the set of similar images is eventually returned to the user. For example, in the context of our experiments, searching for the images containing objects that are similar to the ones in the query typically generates between 50 and 600 consecutive queries that search in a DB that is 50 to 600 times larger. This demanding process increases the impact of the performance problems traditional multi-dimensional index techniques suffer from.

The second contribution of this paper is an initial exploration of the consequences of using local descriptors together with up-to-date database multi-dimensional indexing strategies. While this paper *does not present any new indexing scheme*, it experimentally shows that the three most efficient indexing techniques known today are still too slow to be used in practice with local descriptors. Using today's DB indexing techniques with local descriptors requires to *add-up* the response time of each individual query, which makes the global response time far above what one might tolerate. We therefore list problems and enumerate several potential solutions for building efficient content-based retrieval systems supporting fine-grain image recognition as suggested by modern image processing techniques.

This paper is structured as follows. Section 2 describes the local descriptors. Section 3 overviews multi-dimensional indexing techniques. Section 4 evaluates the performance of the three most efficient indexing techniques known today on a large base of high-dimensional data in the context of local descriptors. Section 5 presents some open issues and an initial set of solutions before concluding.

2 Local Descriptors for Robust Object Recognition

The descriptors we use are an extension to color images of the fine-grain recognition scheme for grey-level images originally proposed by [8] and extensively used and evaluated by [17]. This scheme is highly robust to grey-level image transformations: it detects similar elements in images despite rotations, translations, scalings, partial occlusions, changes of backgrounds or viewpoints, ... Color increase their robustness towards illumination variations. The main goal of this section is to present the original recognition scheme and our extension to color images.

2.1 Grey-Level Descriptors

Computing the grey local descriptors encoding information about a single image is done in two steps. First, specific points in the image, called interests points, are determined such that it is very likely that a point found in one image will also be found in another image that slightly differs from the first one. Schmid showed in [18] that the extractor by Harris [10] had the best behavior. The number of points in one image typically varies between 50 and 600, depending on the shape of its signal. Second, the signal around each interest point is convoluted with a Gaussian function and its nine derivatives up to the third order. These derivatives are mixed together to enforce invariance properties and to make descriptors robust to the changes mentioned above. Translational invariance is due to the fact that descriptors are computed around each interest point, ignoring their respective localization in images. The angle of rotation of images can be algebraically eliminated from the ten derivatives, providing nine resulting quantities invariant to rotations. Among them are the norm of the gradient and the Laplacian of the signal.

It is possible to gain invariance towards illumination variations modeled by $I \mapsto aI + b$. This model, although simple, describes quite accurately what happens when the global intensity of the illumination varies slightly. It is possible to withdraw parameters a and b, resulting in 7 invariants.

Scale invariance can be achieved by adopting a multi-scale approach [6]: the computation of the interest points and their descriptors is repeated at various scales, and all the resulting values are used to describe the image.

2.2 Extension to Color

Using color allows to cope with more realistic illumination variation models. Each pixel of a color image is defined by 3 values which can be coded in many ways. The RGB system was chosen because it facilitates the extension of the grey descriptors to color. 30 derivatives (10 per channel) are now used after having extracted points to characterize the signal.

Rotational invariance is obtained by withdrawing the angle of rotation. 9 invariants are obtained per channel, and two other ones mix the three channels. Photometric invariance is more complex because different illumination models can be considered. A general model is $(r', g', b')^T = M(r, g, b)^T + V$ where M is a 3×3 matrix and V a vector. A very common model

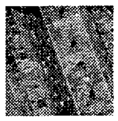
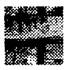
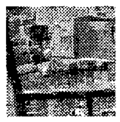
Query	Best Answer	Query	Best Answer
			

Figure 1: Two queries made with image fragments and the most similar images found.

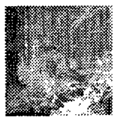
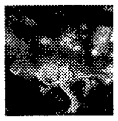
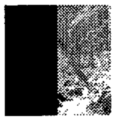
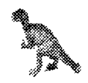
Queries			Answer
			

Figure 2: Three queries with cluttered complex scenes and the retrieved image (on the right).

is obtained when M is diagonal. In this case, the descriptors have 24 dimensions. Coping with spectral variations of the light source requires a full rank matrix M, but the corresponding invariants are known only if no rotational invariance is needed.

2.3 Evaluating the Recognition Capabilities of the Descriptors

Descriptors computed over all the images are inserted into an index, off-line. The similarity retrieval first extracts interest points from the query image and then computes the corresponding local descriptors. Each query descriptor is used to probe the index. The index returns similar descriptors found in the DB, from which it is possible to determine the id of the associated image. It is therefore easy to count the number of time each image id is returned by the index during the whole retrieval process. At the end of this process, the counters allow to rank the candidate images by decreasing similarity.

In general, evaluating the recognition capabilities of descriptors is difficult. The results depend obviously on the intrinsic power of the descriptors, but also on the content of the DB. For example, retrieving sunshines is much easier with a DB containing only sunshines and images of dark tropical forests than with a DB containing sunshines and sunrises. The impact of this subtle side-effect can be limited when the DB stores images specifically chosen to stress a particular aspect of the descriptors. For example, the robustness to illumination variations of the descriptors can be precisely evaluated if the DB includes a series of identical images that differ only by their illumination characteristics. The DB we used to evaluate our color descriptors contains such images in addition to other, real-life, non-specific images.

2.3.1 Evaluation with Grey-Level Images

Schmid's evaluations show the robustness of the descriptors in the context of grey images. The DB used for her experiments was made of about 1000 images: 200 pieces of art, 100 aerial images of the downtown of Marseille (France), and 720 images of 3D objects.

The aerial images are the most challenging. They all "look" similar (roofs do not differ so much) and images are composed of 3D micro structures (chimneys, antennas, cars seen from above, etc.). In her tests, the query images were not part of the DB. The query images were taken during a subsequent pass over the city. This changes the viewpoint (the facades of buildings become visible or disappear) and the composition of images (some cars have moved). The plane, however, took all the images in a short time frame, making unchanged shadows.

To illustrate the recognition power of the descriptors, two examples from Schmid's work are presented. Figure 1 shows the best answer (i.e., the most similar image) when queried by a small image fragment. Queries using fragments provide good results as long as the size of the fragment stays above 10% of the size of the original image. Smaller fragments degrade the results rapidly.

Turning to Figure 2, several images of a dinosaur seen from very different points of view were stored in the DB. Three different query images were then cooked-up using a complex background on which a view (that is not in the DB) of the dinosaur was superimposed. In addition, the left part of the third query image was deleted (only the tail of the dinosaur remains visible). The best answer provided by the system is the image of the dinosaur that is on the right of the Figure (it is the point of view in the DB that is the closest to the one in each query). This result emphasizes the robustness of the method caused by the locality of the descriptors and the by the counting process.

2.3.2 Evaluation with Color Images

To evaluate the recognition power of the color descriptors, we used a 40 Mb DB made of the 24 dimension descriptors derived from 1,816 real-life color images. 1,206 images come from 50 seconds of a video. The remaining images come from a DB of still images found on the web.[1] The total number of descriptors computed from these images is 413,412.

The still image DB is composed of sequences where one or two parameters vary slowly in a controlled

[1] http://www.inrialpes.fr/movi/pub/Images/index.html

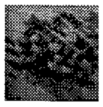
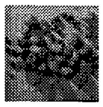
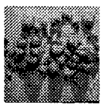
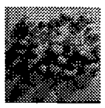
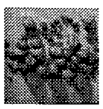
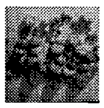
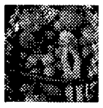
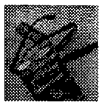
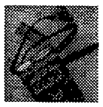
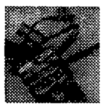
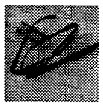
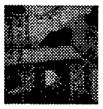
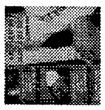
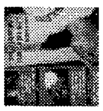
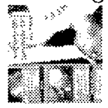
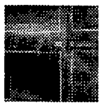
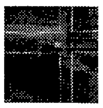
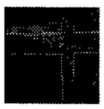
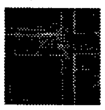
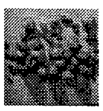
Invariance	Query Image	Answers and Scores				
Query A: main light source motion	 447	 86	 62	 61	 60	 58
Query B: camera motion	 255	72	57	52	37	37
Query C: image composition variation	 334	 247	 166	153	108	 63
		Answers				False Neg.
Query D: illumination variation	518	 346	341	 307	 259	
		Answers				False Neg.
Query E: light source spectral variation	 408	 296	 165	 69	 19	

Figure 3: Illustrating 5 aspects of the robustness of the local descriptors extended to color.

manner: There are sequences with a variation of the intensity of the light source, sequences taken by a rotating or a translating camera, ... This allows to specifically test the robustness of the descriptors and of the retrieval method towards this or that parameter. In Figure 3, the number of descriptors computed for each query image is indicated below each query image. In addition, the number of descriptors that matched is indicated below each DB image returned by the system. Note that the average noise level (i.e., the number of matches whatever the query is) is between 5 and 10. Due to space limitations, we present here only a subset of the results showing the robustness of our invariants and their increased recognition power due to the extension to color. More results are given in [2].

Query A: Invariance to Light Source Motion. We first studied the robustness of our invariants towards the motion of the light source (e.g., the sun which moves during the day). Being robust to that motion enables for example to query the system using a morning image of a landscape and to get evening images of that same landscape. The results in this case mostly depend on the intensity of the shadows. As far as the shadows are not too dark, there is still enough information to compare light and dark objects. Some results are given by the first row of Figure 3.

Query B: Geometric Invariance. In this test, the camera is moving with a very general motion. The sequence has 20 images and we chose the eleventh one as a query. 14 images of this sequence where found as being the most similar to the query image, as partially illustrated by Figure 3, second row.

Query C: Robustness to Changes in Image Composition. In this test we used a sequence of images in which various objects are added or removed. Then, we picked one of these images to be the query, and asked the system to return the most similar images. An example is shown in Figure 3, third row. Images with additional objects are preferred over images with missing objects because they usually allow more matches between descriptors.

Query D: Robustness to Illumination Variations. Robustness towards intensity variation of the main light source was tested using a sequence of 9 images. The first image of the sequence was used as a query, and 7 of the other images were retrieved as the most similar, as illustrated by Figure 3, fourth row. The brightest image of the sequence is a false negative: although it is very similar, it is quite saturated and many descriptors were therefore unusable.

Query E: Robustness to Light Spectrum Variations. An interesting photometric variation occurs when the spectrum of the light source varies. Such variations are correctly handled by a model using a diagonal matrix while the variation remains small. This can be seen on the results partially shown on Figure 3, fifth row. For greater variations, a model with a full rank matrix is needed.

3 Database Techniques for Multi-Dimensional Indexing

The descriptors presented above are robust and well suited to detect similar elements in color images. It is therefore natural to integrate them in a large content-based retrieval system. In this case, DB indexing techniques are needed to speed-up similarity-based searches. Therefore, we now present an overview of the indexing techniques used in databases, traditional approaches first. We then focus on the two strategies that provide today the most efficient support for multi-dimensional searches. We chose these two strategies to built our own system. Their performance, however, is far too slow to make the system usable in practice, as detailed in the next section.

3.1 Traditional Approaches

Still images indexing techniques can be classified in two families: *data-partitioning index methods* that divide the data space according to the distribution of data, and *space-partitioning index methods* that divide the data space along predefined lines regardless to the actual values of data and store each descriptors in the appropriate cell.

Data-partitioning index methods all derive from the seminal R-Tree [9], where bounding regions are rectangles, spheres [21] or both [14]. There are many strategies for merging or keeping separated regions at each level of the tree [5]. [4] demonstrates that, in the general case, using a 50%-quantile to split nodes in overflow leads to the unexpected effect that, during a high-dimensional search, the probability of accessing every page of the index gets close to 1. In this case, the resulting access pattern to disk pages severely hampers the search performance since it is totally random.

Space-partitioning techniques like grid-file [15], K-D-B-Tree [16], LSD^h-Tree [11] typically divide the data space along predetermined lines regardless of data clusters. These techniques are known to become inefficient when the dimension of data gets above 10 to 16 dimensions (see [1, 12]). They also face the problem of indexing large volumes of empty space.

3.2 VA-File and Pyramid-Tree

All the techniques presented above generally work well for low-dimensional spaces. Their performance, however, is known to degrade as the number of dimensions of the descriptors increase. This phenomenon is known as the *dimensional curse.* In other words, navigating within the index becomes more costly than a simple sequential scan in high-dimension spaces when searching nearest-neighbors.

Two innovative approaches, the Pyramid-Tree [4] and the VA-File [20], however, have been recently proposed to tackle head-on the dimensional curse phenomenon: they have been designed specifically such that their behavior does not dramatically degenerate when the data dimension increases. These two strategies provide today among the most efficient support for multi-dimensional similarity search.

The VA-File approach comes from the observation that the brute-force sequential scan proves to be competitive in high-dimensions (it is often the fastest search technique). Therefore, this approach tries to boost the sequential search by eliminating many useless comparisons using rough approximations of descriptors. This method manages two different sets of data: a file storing all the descriptors, and another file storing their geometrical approximations. The performance of this method is at its best when this latter file fits in main memory.

The geometrical approximations are computed using an irregular grid laid over the data space. To build the grid, the method first splits each dimension d_i in 2^{b_i} slices (coded using b_i bits), such that all slices are equally full. All d dimensions are sliced this way. The intersection of slices define 2^b cells, where $b = \sum_i b_i$, numbered from 0 to $2^b - 1$. To fill the index, all the descriptors are then read, and the approximation of a descriptor is given by the cell number into which it falls. The approximation file is small since cell numbers are typically smaller than descriptors. The VA-File is therefore a compression technique.

During a search, the algorithm first uses the approximations to determine which cells can not be part of the result (this filters out all the irrelevant descriptors). The remaining cells are scanned in an increasing order of distance. Starting from the closest cell, the algorithm randomly fetches the associated descriptors and performs distance calculations until n nearest-neighbors are found. The filtering step reduces the number of records to fetch and the number of comparisons and calculations to perform with respect to the traditional sequential scan.

Berchtold et al. propose, with the Pyramid-Tree [4], a method that divides a space $[0,1]^d$ in $2 \times d$ pyramids. The top of each pyramid is placed at the center of the data space. The base of each pyramid has a surface of $d-1$ dimensions. Each pyramid is assigned a different number. Each pyramid is then cut in slices that are parallel to its base. Partitioning the data space this way has the interesting property to create a number of cells that increases linearly (and not exponentially) with the number of dimensions.

Sliced-pyramids enable to map any point of the multi-dimensional space into a pair (pyramid number, slice number). Therefore, a B^+-Tree index can be used instead of a multi-dimensional index structure. B^+-Trees are known to be very efficient for this type of data and for range queries. In addition, they nicely cope with concurrent updates and can be made failure resistant. These two properties are very desirable and often lack to other solutions.

4 Performance Evaluations

The VA-File, the Pyramid-Tree and the sequential scan proved to be efficient in the context of searches with global descriptors. We therefore measured their performance when used together with *local descriptors*. The first experiment shows the performance of the techniques when the dimension of the descriptors increases. The second experiment shows the impact of the size of the database on the response times. Last, the third experiment shows the influence of the (large) number of descriptors forming a single query on the response times. We first describe our experimental setup.

4.1 Experimental Environment and Overview of the Database

We used the source code of the VA-File and of the Pyramid-Tree provided by their respective authors to perform our performance evaluations.[2] We implemented our own version of the sequential search. All the algorithms were ran on a SUN Ultra 5 workstation running SunOS 5.7. Its CPU is a 333 MHz UltraSPARC-IIi, with 384Mb of main memory and 8Gb of local secondary storage. All the response times have been obtained using `getrusage()`.

We analyzed the codes of the VA-File and of the Pyramid-Tree to insert at the appropriate places timer start and stop instructions. We slightly changed the metric used by the Pyramid-Tree to compute the distances between points in the data space: it was L_∞ and we changed it to L_2. Without this patch, the nearest-neighbors returned by the Pyramid-Tree would not have been identical to the ones returned both by the VA-file and by the sequential search. This patch seems to have no significant impact of the response-time.

Our implementation of a sequential search strategy assumes that all the query descriptors fit in main memory.[3] All query descriptors are read at once at the beginning of the search. Then, each descriptor stored in the DB is read sequentially and compared against all the query descriptors. The sets of neighbors for each query descriptor are maintained dynamically. The result is delivered once the end of the DB is reached. This implementation behaves somehow like a join in which the smaller relation is fully fetched before reading tuple after tuple the larger relation.

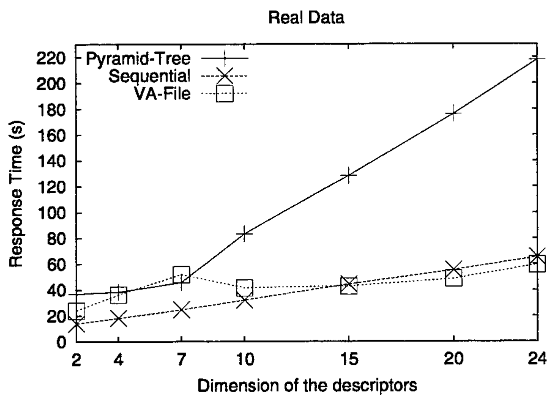
Two databases where created to perform the following performance measurements. The first DB, already described in Section 2.3.2, is made of 413,412 descriptors of 24 dimensions derived from 1,816 color images. The descriptor distribution along each dimension is far from being uniform. For example, the second component varies from about -20 to about 36, and 99% of the values are between -1 and 1. Since many performance evaluations published in the literature assume uniformity, we generated our second DB in which the $24 \times 413,412$ values have been picked between 0 and 1 using a random uniform generator. Queries use images and random vectors that are different from those stored in the databases.

4.2 Varying Data Dimensionality

This first experiment shows the influence of the data dimensionality on the performance of the three techniques we study here. We first computed the 413,412 descriptors having 24 dimensions using real data from the DB described above. These descriptors were then

[2]We are grateful to Roger Weber who graciously gave us his implementation of the VA-File. The source code of the Pyramid-Tree is available on the Web page of Stefan Berchtold (`http://www.stb-gmbh.de/~berchtol/`).

[3]There are on average 150 descriptors per query. Storing them in main memory requires less than 15K in the case of 24 dimensions. Furthermore, if we assume that 10 nearest-neighbors are maintained for each query descriptor, then, about 100K are required to store all these neighbors when 150 descriptors are in the query.



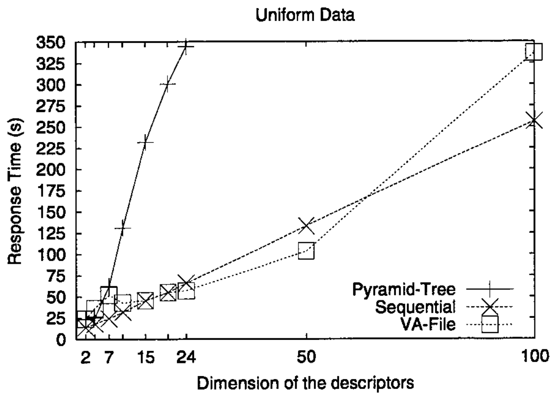


Figure 4: DB storing 413,412 desc., 150 desc. per query, *increasing dimension of descriptors.*

truncated to 2, 4, 7, 10, 15, 20 and 24 dimensions. 413,412 descriptors uniformly distributed have then be randomly generated for the same dimensions, but also for greater dimensions (up to 1,000). In the case of 10 (resp. 24) dimensions, the resulting DB occupies 16,536,480 (resp. 39,687,552) bytes.

Once the DB created, a query containing 150 descriptors was computed using an image outside the DB or new random numbers. We then truncated them to the appropriate dimensions in order to create the requests that will query the real and the synthetic databases. The response times given by Figure 4 are the cumulative response times of 150 consecutive requests, each returning 10 nearest-neighbors.

The performance of the algorithms using real data are illustrated by Figure 4 (left). In this case, the performance of the Pyramid-Tree severely degrades above 7 dimensions. Beyond, its response time gets too big to remain competitive. The VA-File and the sequential search exhibit better performance, and degrade less rapidly when the dimension increases. The performance of the sequential search is linear with the dimension, and searching 150 descriptors among 413,412 takes approximatively 66 seconds in the case of 24 dimensions.

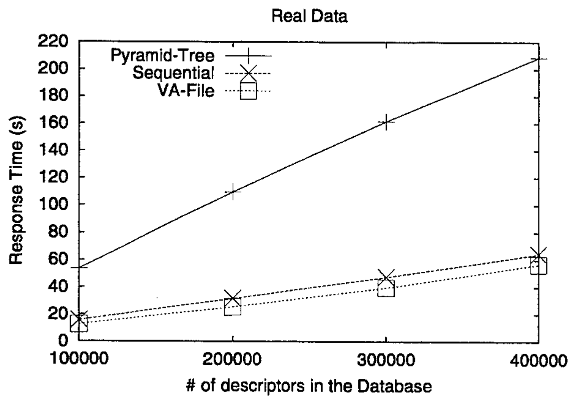
The performance of the VA-File and of the sequential search are rather similar, except when the number of dimensions is small. In this case, for 2 dimensions, a VA-File search takes about 24 seconds, and about 52 seconds in 7 dimensions (25 seconds are needed in 7 dimensions for the sequential search). Few dimensions makes the filtering step poorly selective, and exploiting the approximations in addition to computing many actual distances is part of the observed overhead.

The performance corresponding to the experiments that use uniform data are given by Figure 4 (right). This Figure does not show any response time of searches for data having more than 100 dimensions since they become too high to remain significant. In this Figure, the Pyramid-Tree is again the technique having the worst response time. Below 15 dimensions, the sequential search performs better than the VA-File, for similar reasons as the ones mentioned above. When data has 50 dimensions, the VA-File returns its answer (recall that 150 consecutive queries must be submitted before returning the answer) in about 104 seconds while the sequential search needs 134 seconds. In this case, the VA-File strongly benefits from the geometrical approximations and from its filtering strategy. Above 50 dimensions, the sequential search becomes faster than the VA-File. With 100 dimensions, the sequential search needs 256 seconds and the VA-file 336. These results are confirmed by those given in the article presenting the VA-File, because at this point, the approximation file becomes too large to fit in main memory, increasing the number of I/Os and the overall response time.

Regardless of the nature of the data stored in the DB (real or uniform), the response times needed to search the 10 nearest-neighbors of 150 descriptors in a rather small DB are big: around a minute for both the sequential scan and the VA-File with 24 dimensions. These response times are above what one might tolerate if these techniques were part of a real system. The next experiment investigates further the influence of the size of the DB on the response times.

4.3 Varying Database Size

Figure 5 shows the impact of the database size on the response times of the 3 techniques. For this experiment, we reused the 413,412 descriptors previously computed with 24 dimensions, and generated new databases by keeping only 100,000, 200,000, 300,000



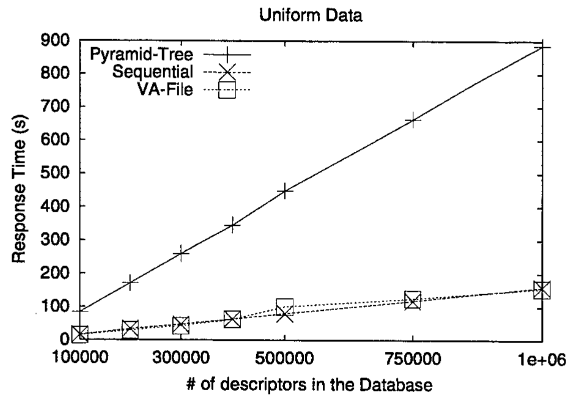


Figure 5: 24 dimensions descriptors, 150 desc. per query, *increasing the size of the DB.*

and 400,000 of them. The requests are made of the same 150 descriptors in 24 dimensions as above. We could not easily create larger databases since the amount of real data we could use was limited. It is easy, however, to create uniform databases of arbitrary sizes. We therefore created such databases, and the larger we generated contains 1,000,000 descriptors (96Mb), and this could correspond to more than 6,500 images if we assume that an image is described by 150 local descriptors on average.

Figure 5 (left) shows the case with real data. What was observed in the previous experiment can be found here again. That is, the Pyramid-Tree is more expensive than other techniques, and that the VA-File is slightly better than the sequential. Still, 15 seconds are needed to perform a search of a DB made of only 100,000 descriptors.

Figure 5 (right) shows the case with uniform random data. For a DB made of 1,000,000 descriptors the response time for the sequential scan or for the VA-File is of about 3 minutes (160 seconds). This result clearly forbids the use of these techniques in a real system indexing millions of images (and therefore many more descriptors). Furthermore, our request has only 150 descriptors, and they are, in the general case, more numerous. The next experiment focuses on this problem, and shows how the number of descriptors per request influences the response times.

4.4 Varying Query Length

All the descriptors used in this experiment have 24 dimensions. To generate the queries used here, we searched in our real DB images for which 100, 200, 300 and 400 descriptors were computed. We also made-up an artificial query that has only one descriptor since this is the typical case for which the database techniques have been designed for. Forging synthetic queries is trivial, and the largest one had 1,000 descriptors.

The results of this experiment given by Figure 6 show again that only the VA-file and the sequential scan remain interesting. The response time, however, rapidly grows with the number of descriptors in the query. For example, 400 real descriptors cause the response time to jump to 121 seconds for the VA-File and to 185 for the sequential. With uniform data, 997 and 1,042 seconds are needed respectively for the VA-File and the sequential with 1,000 descriptors.

The number of descriptors in each query is directly related to the number of interest points detected in the query image (see Section 2). This number can clearly be very big depending on the image and on the detection strategy. It is crucial that the cost of a query having many descriptors does not increase so fast, as illustrated by this experiment.

5 Conclusion and Perspectives

The dimensional curse phenomenon makes existing multi-dimensional indexing techniques barely efficient when content-based retrieval is performed on a (traditional) global similarity criteria. Fine-grain image recognition, supported by local descriptors, magnify even more this phenomenon. While performance problems are clearly seen in our experiments, worse results are expected if the techniques were used in a more realistic environment, where the size of the image bank is far bigger than our database (up to several Gb), where the descriptors have many more dimensions (hundreds) or where the number of descriptors used for one query is much greater (thousands).

To fully exploit the power of fine-grain image recognition, it is therefore crucial to come-up with new indexing techniques specifically designed to efficiently support the use of local descriptors. We therefore list

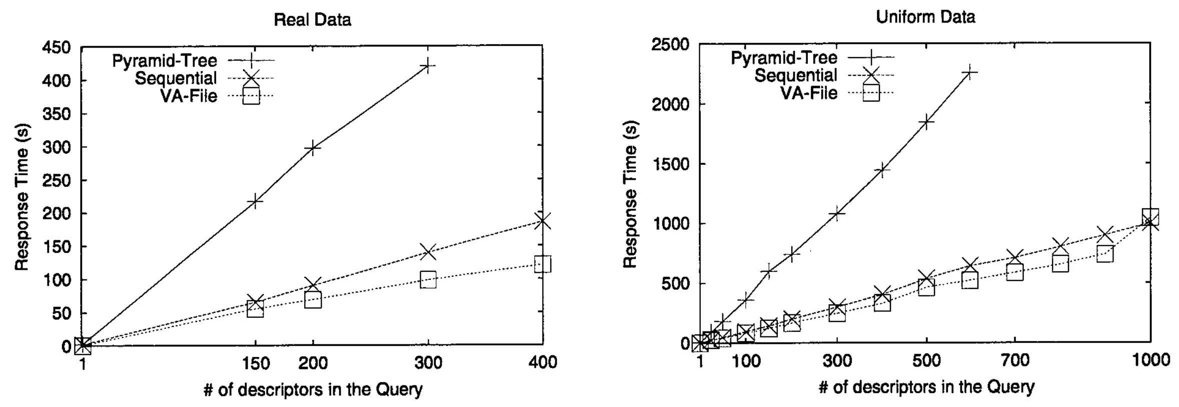


Figure 6: DB storing 413,412x24d descriptors, *increasing the number of descriptors in each query.*

in the following several directions for future investigations aimed at tempering the above mentioned effects.

Numerous local descriptors for a single query creates redundancy. When local descriptors are used, recognition is based on multiple consecutive searches, each returning information which, once accumulated and post-processed (cross-checking), gives the final answer. Some images stored in the database will belong to this final answer because several descriptors of the query matched with several descriptors associated to these images. There is therefore a certain form of redundancy in the information used during the complete search process (because all these query descriptors are associated with a *single* query image) and in the information returned (because an image is found similar since many of its descriptors match). It is possible to use this redundancy in (at least) two ways.

First, the search process can be restricted such that it checks only the descriptors that are in the same cell (or page, ...) than the one in which each query descriptor falls.[4] This avoids the typical and mandatory lookup of all neighboring cells during nearest-neighbors searches, which is known to be expensive since many cells must be visited. If the search process returns, for *each* query descriptor, only the nearest-neighbors that are *in the same query cell*, and ignores other potential neighbors that are in adjacent cells, then the search cost would be reduced. The result of each query, however, is clearly a rough approximation of what would be returned if the normal search process was enforced. The quality of this approximation is improved as the time goes by since many query descriptors are used to get the images that are similar to *one* image. This strategy has the interesting property to trade accuracy (of the final result) for efficiency, yet, cross-checking what is returned by each individual search is a natural way to consolidate the final answer and fully uses the observed redundancy.

[4]It is unlikely that *all* query descriptors fall in empty cells. If too many empty cells are found, then the search can switch-back to its regular behavior.

Another way is to stop the search before having used all query descriptors. In this case, the search is greedy, and each partial result returned by each individual query is immediately processed and updates the (in-progress) final result. When this current (not yet complete) final result has a high probability to be the complete final answer, the search is stopped, the remaining query descriptors are skipped, and the result is returned to the user.

Both strategies can be combined to search only the relevant cells (ignoring adjacent cells) using a limited number of query descriptors.

Exploit the distribution of data to accelerate the queries. Not all descriptors carry the same amount of information: some are associated to many images, some others are rare. Therefore, the matching of two descriptors returns a more or less discriminative information, making the associated database image to be more or less likely part of the final result. In this case, a Bayesian formalism may help in determining the probability for each match to help converging towards the final result. In addition, it is possible to sort the descriptors in the query such that it starts with the descriptors that are the most informative. It is therefore possible to stop the search as soon as the probability of having the final answer is high enough, or as soon as the search starts using the descriptors that do not help converging. This technique has the interesting property to refine the search as the time goes by. In addition, the search accuracy can easily be made controllable by the user.

Change the management of memory to benefit from consecutive queries. Traditional techniques assume that a single search within the database is sufficient to return the final answer. Therefore, what is fetch in memory during a search benefits to the next query only by chance: if the second query is lucky enough to use some of the data brought in memory by the first query, then its response time is enhanced because some data is already cached. A better mechanism can be designed when local descriptors are used. In this case, we know in advance that a large amount of consecutive queries will be submitted to the database. Therefore, it may be interesting to pick the next query descriptor with respect to what is already in the cache. That is, the next descriptor used to query the database can be the one which is the most likely to have its nearest neighbors *already in memory*, brought in by previous queries. Therefore, instead of consuming all the query descriptors sequentially as the natural search process does, descriptors are picked in a memory conscious way. In addition, since all the query descriptors are known before hand, prefetching might be used to remove cache misses from the critical path.

Use several low-dimension indexes instead of a unique high-dimension index. It is known that the cost of content-based retrieval grows fast when the dimension of data increases. It is therefore potentially interesting to evaluate if querying many low-dimension indexes instead of a unique high-dimension index gives good results. These "small" indexes must be constructed in such a way, and their use must be such that the result they return is identical to what would return a regular index. A query would then have to be transformed in multiple sub-queries, each interrogating a given (small) index, possibly in parallel.

This scheme tries to limit the problem of dimensionality curse by enforcing multiple interrogations of low-dimension data for which efficient indexing schemes exist. On the other hand, additional processing steps are needed, especially in the case of a nearest-neighbor search. [3] is an initial investigation of this idea, limited to the case of range queries.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD*, 1998.

[2] L. Amsaleg and P. Gros. Content-based retrieval using local descriptors: Problems and issues from a database perspective. *Pattern Analysis and Applications*, 4(2/3):108–124, 2001.

[3] S. Berchtold, C. Böhm, D. Keim, H. Kriegel, and X. Xu. Optimal multidimensional query processing using tree striping. In *DaWaK*, 2000.

[4] S. Berchtold, C. Böhm, and H. Kriegel. The Pyramid-Tree: Breaking the curse of dimensionality. In *ACM SIGMOD*, 1998.

[5] S. Berchtold, D. Keim, and H. Kriegel. The X-tree : An index structure for high-dimensional data. In *VLDB*, 1996.

[6] Y. Dufournaud, C. Schmid, and R. Horaud. Matching images with different resolutions. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, June 2000.

[7] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3, 1994.

[8] L. Florack, B. Romeny, J. Koenderink, and M. Viergever. General intensity transformation and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2), 1994.

[9] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD*, 1984.

[10] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[11] A. Henrich. The LSD^h-tree: An access structure for feature vectors. In *ICDE*, 1998.

[12] A. Hinneburg and D. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB*, 1999.

[13] J. Huang, S. Kumar, M. Mitra, W. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, 1997.

[14] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *ACM SIGMOD*, 1997.

[15] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM TODS*, 9(1), 1984.

[16] J. Robinson. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In *ACM SIGMOD*, 1981.

[17] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. IEEE *Trans. on Pattern Analysis and Machine Intelligence*, 19(5), 1997.

[18] C. Schmid, R. Mohr, and Ch. Bauckhage. Comparing and evaluating interest points. In *Proc. of the 6th Int. Conf. on Computer Vision*. IEEE Computer Society Press, 1998.

[19] M. Stricker and M. Swain. The capacity of color histogram indexing. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, 1994.

[20] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.

[21] D. White and R. Jain. Similarity indexing with the SS-tree. In *ICDE*, 1996.

A Semi-Automatic Object Extraction Tool for Querying in Multimedia Databases*

Ediz Şaykol†, Uğur Güdükbay and Özgür Ulusoy
Department of Computer Engineering, Bilkent University
06533 Bilkent, Ankara, Turkey

{ediz, gudukbay, oulusoy}@cs.bilkent.edu.tr

Abstract

Considering the complexity and huge volume of image and/or video data, efficient methods need to be developed for querying multimedia databases. A well-known technique used for querying multimedia data is query-by-feature (e.g. color, shape, texture, size) of the objects residing in images and/or video frames. In this paper, we propose a tool for extracting objects from images and/or video frames, called *Object Extractor*, as well as the ways of coping with the object features within extracted objects. The tool is semi-automatic in the sense that the user specifies the colors on the object by clicking the mouse to make the tool capture object pixels. In order to extract objects, an improved version of the *Flood Fill* algorithm for polygon filling is provided. The extraction algorithm uses filtered images to perform better. Moreover, the experimental results obtained for evaluating the performance of the tool in extracting objects are presented. It is shown through these results that a few mouse clicks would suffice to extract objects effectively.

Keywords: Object extraction, flood filling, color median filtering, color space transformations, color quantization.

*This work is supported by Turkish Scientific and Technical Research Council (TÜBİTAK) under the grant number EEEAG-199E025.

†Author for Correspondence

1 Introduction

Advances in multimedia technology accelerate the amount of digitized information so as the data stored as image and video content. Both of these data types require application-dependent processing strategies and easy-to-handle storage methods when stored in a database. As far as the querying process of image and video databases is concerned, spatial (for both image and video data) and spatio-temporal (for video data only) as well as semantic information are taken into account. Semantic querying requires more sophisticated techniques that encode the semantic meaning of the image and/or video content. Spatial and spatio-temporal querying necessitate a query pre-processing phase in which the objects and their corresponding features (e.g. color, shape, texture, size) are extracted. The motivation of the tool presented in this paper is to facilitate the pre-processing phase of the query-by-feature sub-system of the video database querying system being developed at Bilkent University [7].

There exist a considerable number of multimedia data querying systems in the literature [5, 9, 13, 18]. The pre-processing phase in most of these systems includes object extraction in order to figure out the hidden object-based information from the image and video data. Based on the type of user interaction in the pre-processing phase, the object extraction methods can be grouped into three categories:

Fully Automatic Extraction Methods: In these methods, the extraction process for image and/or video data is performed automatically. Since the whole process lacks user interaction, not all types of images and video can be handled with this approach. The QBIC system [9] employs a method based on *Foreground/Background* approach for fully automatic object extraction. This method processes only images and video frames having a separable background [1]. Jain and Vailaya employ edge-detection algorithms for extracting object boundaries from images [11]. Their method for edge-detection is the famous *Canny Edge-Detection* algorithm [3]. Chang et al. describe automatic feature extraction methods for color, texture and shape features of images in [4].

Semi-Automatic Extraction Methods: In these methods, the user assists the object extraction process. One way of assistance is to facilitate the object extraction of an image via clicking on the object pixels to visualize the object area. *Flood Fill* algorithm for polygon filling [10] may be adopted to determine the pixels in the object area for extraction process. Another semi-automatic method is based on the *snakes* concept of computer vision [12]. In this method, the user specifies a bounding polygonal region for an object and the object boundaries are determined from this region automatically. The QBIC system uses these techniques for the object extraction process [1]. The range of images and/or videos handled by these types of methods is larger than that of fully automatic methods but there still exist some types of data that need more user assistance for a proper object extraction.

Manual Extraction Methods: In these methods, the user-computer relation is more than interaction, because the user manages all the extraction process. For example, in order to encapsulate an object region with a (minimum) bounding polygon, the drawings of the users will be used as is. Obviously, any type of image and video data can be handled manually since any kind of data is perceptually comprehensible to the user. However, the manual extraction is a very tedious process and cannot be applied to very large datasets.

The more the user participates in the extraction process, the less the image restrictions and requirements for proper object extraction are needed. The basic aim in object extraction is to minimize the user interaction throughout the extraction process without discarding any type of image or video data. Thus, a powerful extraction system should include tools for each of the above extraction methods not only to extract objects but also to extract the corresponding object features.

The *Object Extractor* tool proposed in this paper extracts objects in images and/or videos semi-automatically. In order to increase the quality of the processed images and/or video frames, color space transformations, quantization and color filtering are employed before processing the input. The filtered input leads to an increase in the performance of the object extraction algorithm. *Flood Fill for Extraction (FFE)* algorithm is employed for extracting objects and as a result of the filtering steps, the tool provides an environment for processing images and/or videos effectively.

The organization of the paper is as follows: Section 2 summarizes the techniques related to the principles of our *Object Extractor* tool. Sections 3 explains the design of *Object Extractor*, Section 4 describes the object extraction algorithm. Section 5 presents the experimental results obtained for evaluating the performance of the tool. Finally, Section 6 concludes the paper.

2 Background Information

One of the most important features of objects in image and video data is *color*. Each pixel in an image has a three-dimensional color vector and different color spaces encode color information based on different approaches. The most famous color space model is the *Red-Green-Blue Model (RGB)* where the color vector of a pixel p is the compound of red, green and blue channels $v_p = (r, g, b)$. Another color space model is the *Hue-Saturation-Value Model (HSV)* that is based on color descriptions rather than individual color components and makes the model unique among other color space models $v_p = (h, s, v)$. The *RGB* model has a major disadvantage: it is not perceptually uniform. Therefore, most of the systems use color space models other than *RGB*, such as *YIQ* [10].

The color regions are perceptually distinguishable to some extent. The human eye cannot detect some little color differences and may perceive these very similar colors as the same color. This leads to the *quantization* of color, which means that some pre-specified colors will be present on the image and each color is mapped to some of these pre-specified colors. One obvious consequence of this is that each color space may require different levels of quantized colors, which is nothing but a different quantization scheme. In Figure 1, the effect of color quantization is illustrated. Figure 1 (a) is the original image with RGB color space and (b) is the image produced after transformation into HSV color space and quantization. A detailed explanation of color space transformations (from RGB into HSV) and quantization can be found in [17].

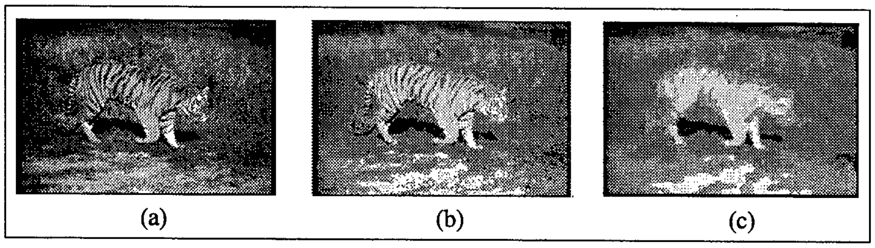


Figure 1: Transformation, quantization and color median filtering of an image. (a) Original image. (b) Image produced by applying RGB to HSV color transformation and quantization. (c) Image produced after applying color median filtering.

Moreover, not all the colors in the image are dominant. Dominance is in the sense that some of the colors may reside in a region relatively small than the others. The *color median filtering* technique [14], a famous method for neighborhood ranking, eliminates these non-dominant colors and produces a filtered image (Figure 1(c)). This technique facilitates the object extraction process because it also eliminates the noise of the color on the object boundaries to some extent.

3 Design of Object Extractor

3.1 Overall Architecture

The *Object Extractor* tool extracts objects from images and videos semi-automatically with the help of the improved version of the flood fill algorithm. The overall architecture of the tool is shown in Figure 2. Videos are segmented with *Fact Extractor* in the system and keyframes of the videos are processed as images. An image is passed through quantization and color median filtering steps and then the final image, where the object is to be extracted, is produced. This filtered final image is processed with *Flood Fill for Extraction Algorithm* to extract the objects along with their features. Since we deal with realistic images and videos in the system, automatic extraction of objects and features would be inadequate, so that the user intervention becomes inevitable. The last step in the process is storing the features of the extracted objects in the object feature database.

Thus, *Object Extractor* is one of the basic tools that cooperate with the query-by-feature sub-system of our video database and querying system [7].

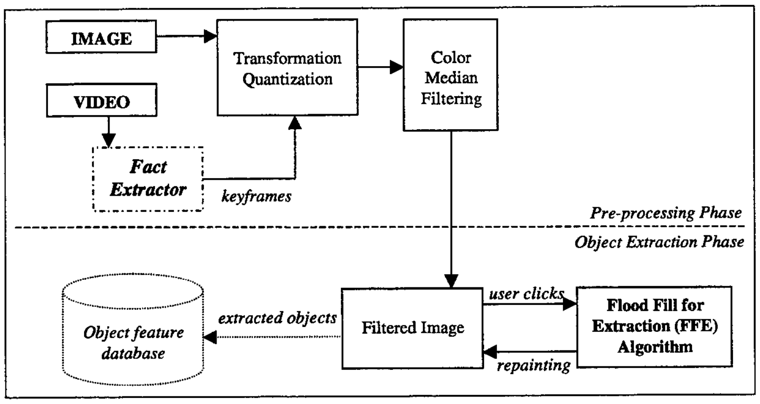


Figure 2: Overall architecture of *Object Extractor*

The image whose objects to be extracted passes through a color space conversion step and the color vectors of all of the pixels are transformed from RGB color space into HSV color space. Then, this data is used in the quantization step yielding 18 hue, 3 saturation, 3 value levels and 4 gray levels. The color quantization step employed here is very close to the one proposed in $VisualSEEk$ [18] and $VideoQ$ [5]. After completing this step, the median filtering algorithm is applied to eliminate the non-dominant color regions.

The *Flood Fill for Extraction (FFE)* algorithm, an improved version of the flood fill algorithm, is designed to specify object regions. The color of the user-clicked pixel initiates the process and it forks into four branches corresponding to the four neighboring pixels (north, east, south and west). As soon as the difference between the processed pixel's color and the initiative pixel's color exceeds a pre-specified threshold, the execution stops. The user may continue to specify new initiative pixels as necessary to extract the object.

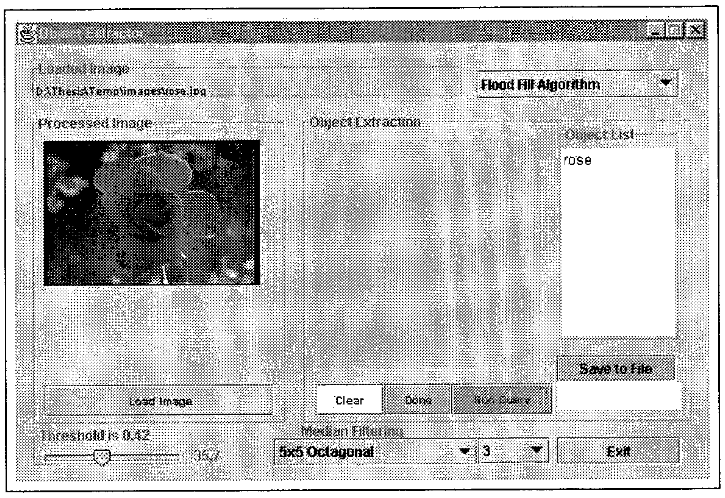
In the pre-processing phase of the *Object Extractor*, color space conversion and color quantization are applied to the input, thus the FFE algorithm performs better. This yields an increase in the effectiveness of the technique. Moreover, since the objects are extracted separately, images containing more than one image are handled successfully. The *Object Extractor* provides an environment where a wide range of images and/or videos are handled effectively by the help of the adapted and improved techniques.

3.2 The User's Assistance

Due to the semi-automatic nature of *Object Extractor*, the user assists the extraction process. The tasks of the user are the identification of the colors in the extracted object and then labelling the object. An appropriate indexing scheme can be adopted for these labelled objects and the indexed data can be queried for the extracted object features such as color and shape. Faloutsos et al. propose an effective querying methodology that is based on quadratic color histogram distance considering the perceptual similarity of colors via cross-correlation [8].

Definition 1 *(t-neighborhood) The t-neighborhood of a pixel p with respect to color is a contiguous set of pixels t_p, where the Euclidean distance between color vectors of p and the pixels on the line segment pp_i, such that $p_i \in t_p$, is not greater than a color difference threshold value t. It is obvious that $p \in t_p$.* □

In the current implementation of our tool, the user clicks on a pixel p_c on the image and the FFE algorithm is initiated with the pixel p_c and the current color difference threshold t. During this execution, the pixels in t_p are repainted for p_c. However, if the object bounds unprocessed pixels, the user may click onto another pixel for extracting other parts of the object. Having satisfied with the extracted object region, the user labels the object.



D:\Thesis\Temp\images\rose.jpg

rose

Threshold is 0.42

35,7

5x5 Octagonal

3

Figure 3: The user interface of *Object Extractor*

3.3 The User Interface of Object Extractor

The user interface of *Object Extractor* has been developed in *Java* programming language and it provides the following functionalities. First of all, since median filtering alleviates color quantization on the image and filters out the non-dominant color regions, the user can see the effects of the median filtering algorithm separately in the tool. This gives the opportunity to the user to decide whether to use median filtering or not. Based on this decision, the color transformation and quantization steps produce a better image. The default color difference threshold is 0.4, which is determined by a reasonable number of experiments. The user interface of the Object Extractor tool is shown in Figure 3.

Moreover, the main usage of this tool is to extract objects for the query-by-feature subsystem of the rule-based video querying system that we develop [7]. As seen in Figure 3, 'run query' button activates this querying operation with the previously extracted objects. The image to be queried is also segmented with this tool and the objects are extracted. Since all of the extracted objects are handled with a proper indexing mechanism, the extracted objects of the query image can be queried with the existing objects. The details of the querying methods can be found in [16].

4 The Object Extraction Algorithm

The *Object Extractor* tool processes both images and/or video frames. The method it employs for both types of data is very similar since each video frame can be treated as a single image. The *Fact Extractor* tool inside the video database system handles video data and produces video keyframes. Thus, videos can be processed in the *Object Extractor* tool through their keyframes.

```
procedure FloodFillforExtraction(Pixel p)
// INPUT: a single pixel p
// the INITIATIVE_PIXEL is global to the method and
// it holds the user-clicked pixel

1.   if (pixelProcessed(p))
2.       return;
3.   endif
4.   setProcessed(p);
5.   if (thresholdPassed(p, INITIATIVE_PIXEL))
6.       paint(p);
7.       FloodFillforExtraction(left(p));
8.       FloodFillforExtraction(right(p));
9.       FloodFillforExtraction(up(p));
10.      FloodFillforExtraction(down(p));
11.  endif
endprocedure.
```

Figure 4: Flood fill for extraction (FFE) algorithm

As discussed above, *Flood Fill for Extraction (FFE)* algorithm works with a transformed, quantized and median filtered image. When the algorithm halts, it repaints some of the pixels on the image and the user may continue the extraction as many times as he/she wants. The pseudo-code of the FFE algorithm is given in Figure 4. When the user clicks on a pixel in order to initiate the algorithm, this pixel is stored in the $INITIATIVE\_PIXEL$ and used globally in the procedure. Line 1 checks the stopping condition and the lines $4 - 11$ correspond to the recursive part. Each pixel is processed only once due to the `if`-statement at the beginning. Since a pixel may be visited more than once, this fastens the algorithm significantly. On the other hand, the test for the threshold in line 5 is performed by evaluating the Euclidean distance between color vectors of the two pixels, namely p and $INITIATIVE\_PIXEL$. If the test succeeds, the pixel p is repainted and the algorithm calls itself recursively for the neighboring four branches. The whole process stops when there is no executing branch in the recursion tree.

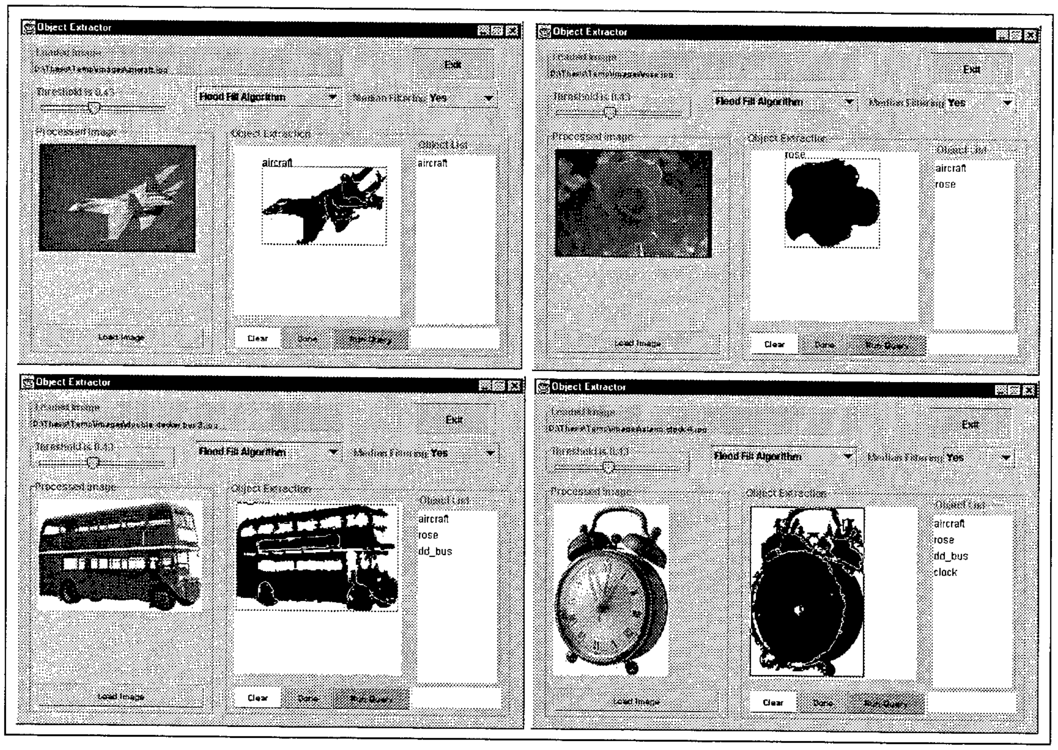


Figure 5: Experimental snapshots for four images sampled in *Object Extractor*

5 Experimental Results

The experiments to evaluate the performance of the *Object Extractor* tool are conducted with the images from *Berkeley University Blobworld Project* [2] and *CoffeeCup Software Photo Gallery* [6]. In the experiments, color median filtering is enabled with 5x5 box filters and applied three times to improve smoothing. The results are presented in Table 1 where each row shows the number of mouse clicks during extraction of an object with different threshold values. A detailed analysis and evaluation of *Object Extractor* with various median filters as well as comparisons with some of the existing object extraction methods and photo editing tools using similar techniques in terms of effectiveness can be found in [15].

The tool gives promising results when the objects are on a separable background in the image. This restriction is inevitable but softened with quantization and color median filtering. This lies in the observation that median filtering facilitates the separation of background from the foreground of the objects. However, the tool performs better in extracting objects containing noise or non-uniformity on the boundaries as well as objects containing holes since it is usually agreed that automatic object extraction tools have difficulty in extracting such objects. Moreover, most of the objects are extracted with a few clicks in different color regions of an object and the semi-automatic nature of the tool

Table 1: Objects versus color difference threshold *t*. *X* means that the repainted region is larger than the object region when extracted with the threshold.

	Number of Clicks		
Object	`t=0.21`	`t=0.42`	`t=0.56`
Rose	3	1	X
Aircraft	5	2	X
DDeckerBus	7	3	1
AlarmClock	8	2	2
BlackGirl	8	2	X
WorldMap	9	2	X
Tiger	3	1	X

does not have a considerable effect on the speed of the extraction process.

6 Conclusion and Future Work

The *Object Extractor* tool is a semi-automatic tool used for extracting objects from image and/or video data. In our video querying system, the queries that specify object features is processed with the help of this tool. The extracted object features are basically the color content and the shape information (in fact the boundary) of the objects. For the former, the color content of the whole image can be stored in order to respond to image-based color queries. For query-by-shape, the boundary information of the extracted objects can be stored as well as some other shape features such as turning angles, area, center coordinates, etc. The operations necessary for query-by-feature sub-system and the improvements on object extraction tool are going to be performed in parallel. One possible improvement in *Object Extractor* will be enabling the users to specify boundary polygon rather than clicking onto the object pixels.

Along with the described work, we have also studied extracting objects automatically not only from images but also from video frames. Since disabling user interaction throughout the object extraction process requires the use of more automatic image processing methods on the images and/or video frames, this is a relatively hard task to achieve. Adopting appropriate indexing structures onto the extracted objects based on the features that are stored in the object feature database is another ongoing project for our rule-based video database and querying system.

References

[1] J. Ashley, R. Barber, M. Flickner, J. Hafner, D. Lee, W. Niblack, D. Petrovic. Automatic and semi-automatic methods for image annotation and retrieval in QBIC. *Proceedings of SPIE-Storage and Retrieval for Image and Video Databases III*, Vol. 2420, 24–35, 1995.

[2] Berkeley University Blobworld Project Start Images, Berkeley University. http://elib.cs.berkeley.edu/photos/blobworld/start.html.

[3] J. Canny. A Computational Approach to Edge-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, 679–698, November 1986.

[4] S.F. Chang, J.R. Smith, H. Wang. Automatic Feature Extraction and Indexing for Content-Based Visual Query. *Technical Report CU/CTR 414-95-20*, Columbia University, January 1995.

[5] S.F. Chang, W. Chen, H.J. Meng, H. Sundaram, D. Zhong. VideoQ: An Automated Content Based Video Search System Using Visual Cues. *ACM Multimedia'97 Conference Proceedings*, 313–324, Seattle, WA USA, 1997.

[6] CoffeeCup Software Photo Gallery. http://www.coffeecup.com.

[7] M.E. Dönderler, Ö. Ulusoy, U. Güdükbay. A Rule-Based Approach to Represent Spatio-Temporal Relation In Video Data. *ADVIS'2000 Proceedings*, LNCS Vol. 1909, T. Yakhno (Ed.), 409–418, 2000, Springer-Verlag.

[8] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petrovic, R. Barber. Efficient and Effective Querying by Image Content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.

[9] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petrovic, D. Steele, P. Yanker. Query by Image and Video Content: The QBIC System. *IEEE Computer Magazine*, 28(9), 23–32, September 1995.

[10] D. Hearn, M.P. Baker. Computer Graphics. Prentice Hall, Inc., New Jersey 1994.

[11] A.K. Jain, A. Vailaya. Image Retrieval using Color and Shape. *Pattern Recognition*, 29(8), 1233–1244, August 1996.

[12] M. Kass, A. Witkin, D. Terzopulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 321–331, 1998.

[13] A. Pentland, R.W. Picard, S. Scarloff. Photobook: Tools for Content-Based Manipulation of Image Databases. *Proc. Storage and Retrieval for Image and Video Databases II*, Vol. 2,185, 34–47, SPIE, Bellingham, Washington 1994.

[14] J. Russ. The Image Processing Handbook. CRC Press in cooperation with IEEE Press, 1999.

[15] E. Şaykol, U. Güdükbay, Ö. Ulusoy. A Semi-Automatic Object Extraction Tool for Storage and Retrieval in Multimedia Databases. *journal paper in preparation*, 2001.

[16] E. Şaykol. Web-Based User Interface For Query Specification in a Video Database System, M. Sc. Thesis, Bilkent University, September 2001.

[17] J.R. Smith, S.F. Chang. Tools and Techniques for Color Image Retrieval. *IS&T/SPIE Proceedings*, Vol 2670, In: Sethi, I.K., Jain, R.C., eds., *Storage&Retrieval for Image and Video Databases IV*, 426–437, February 1996.

[18] J.R. Smith, S.F. Chang. VisualSEEk: A Fully Automated Content-Based Image Query System. *ACM Multimedia'96 Conference Proceedings*, 87–98, NY 1996.

A Data Model for Querying Wavelet Features in Image Databases

Simone Santini* Amarnath Gupta†

Abstract

Multimedia databases deal with storage and retrieval of complex descriptors of image contents, called features. Traditional techniques consider features as "black boxes," often represented as vectors for which the only operation defined is distance computation. This "modus describendi" resulted in the widespread utilization of similarity queries, which rest solely on the computation of distances between feature vectors.

The capacity to express queries more complex than simple similarity, however, rests on the possibility of describing and manipulating the structure of the features. In order to achieve this, features should be described as complex data types inside the database, and opportune operators for manipulating these data types should be defined.

In this paper we try to demonstrate the efficacy and feasibility of such a program by modeling a widely used image feature: the wavelet transform. We represent features using a complex data type derived from arrays, and propose a basic algebra from which operators can be derived to manipulate and query wavelet features.

1 Introduction

Several factors distinguish multimedia database systems from relational or complex object databases. Specifically relevant to this paper is the following. Unlike a traditional database, the tasks of storage and retrieval in a multimedia database are not performed on the original "media objects", but on a set of derived objects called *features*, each produced by applying a series of transformations to the media objects. Most often, regardless of the set of transformations applied, a feature is modeled as a "vector", and the only way to use this vector is through a distance function that compares two vectors and returns a distance [?, ?, ?]. The choice of making vector comparison the sole mode of image retrieval has had mixed benefits. On the one hand, it has led to the discovery of many effective features that work remarkably well in retrieving images with specific characteristics. However, it has also has a negative effect on the data management aspect of the problem. Most of the database-oriented research has been to develop improved query languages and index structures for similarity-search, and for performing result ranking in the presence of feature-weights. A specific area that has remained underexplored is that the database almost never utilizes the internal structure of a feature. For example, while a histogram is a very popular class of feature (used in various forms such as color histograms, co-occurrence matrices, pattern spectra, image shape

*Praja, Inc. San Diego, CA, ssantini@praja.com
†San Diego Supercomputer Center, gupta@sdsc.edu

spectra etc.), a typical multimedia database would often treat a histogram as an opaque object that can only be compared to another histogram, but would usually not interpret it as an array on which patterns can be sought. Thus few image databases can address a query like: "find other images having histograms that have a sharper peak in this range of values, and a deeper valley in this other range" [?].

The ability to express queries other (and possibly more complex) than similarity queries can be achieved when the data models for image database systems can capture the semantics of the features (e.g., two histograms with different number of bins can be compared by coarsening one), and are equipped with a larger set of operators to manipulate the feature structures.

The purpose of this paper is to demonstrate the efficacy of such a data model using a popular and complex feature—the wavelet transform [?]. The model assumes that a suitable wavelet transform has already been computed on an image. It offers a method to represent the resulting feature in a manner that enhances the ability to manipulate the model and retrieve images by complex retrieval operations on this feature. This enables us not to precompute all possible features when the images are inserted, but compute a primary set of features (such as the wavelet coefficients at k levels), and compute a more targeted feature to suit the query at run time. As an example, consider a query that attempts to retrieve all images that have a texture pattern, and the feature database stores significant wavelet coefficients at multiple bands. If the texture pattern of the query image is found to be most dominant in a few of the LH wavelet bands and not in the others, it may be more efficient to select out only the qualifying LH bands to perform the distance computation rather than using a predefined feature over all bands. We illustrate however, that our model does not preclude the precomputation of any feature—in fact, it can express any similarity query based on wavelet features that have been defined in literature.

2 Preliminaries

Our data model for retrieval of wavelet features is based upon the concept of complex objects developed extensively in database literature. A complex object model assumes that a piece of data can be either an uninterpreted base type T such as integer and Boolean, or it can be an instance of an abstract data type, created by type constructor functions. Typical type constructors are *sets*, *lists*, and *tuples*, denoted respectively as $\{T\}$, $[T]$ and $\times_{T_i}$, where T is any base or derived type and the tuple operation creates a "record" data type where each attribute ai may have a different data type t_i. Our data model extends an existing data model developed by Libkin, Machlin and Wong for array objects (henceforth the "LMW" model [?]) which is itself based on an extension of the Nested Relational Calculus. The LMW model considers arrays as functions from a rectangular index set to a specific data type (the *data type of the array*; arrays can be created from other arrays by specifying a function based upon a minimal set of operators. Specifically, in addition to set and tuple manipulation operators, the LMW model allows arithmetic operations on natural numbers and four operations to generate and extract an element out of natural number indexed k-dimensional arrays $[\![T]\!]_k$. The array algebra of LMW is based on three operations:

- The array constructor $[\![e | i_1 < e_1, \ldots, i_k < e_k]\!]$, where $i_h \in \mathbb{N}$, $e_h \in \mathbb{N}$ for all h, and e

is an expression generating values of type t. The constructor builds the array whose value for indices $i_1 \cdots i_k$ is $\lambda e.i_1 \cdots i_k$.

- The subscript function $e[i]$, with $i \in \mathbb{N}^k$, which computes the value of the array function for a given index.
- The dimension function $dim_k(e)$, which returns a list of all the dimensions of the array, and the derived functions $dim_{k,i}(e)$ which return the ith dimension of he array.

A fourth operation, for transforming an indexed set into an array, will not be used in this paper. The model additionally allows any Boolean-valued predicate (of the form *if* $f(x) \circ g(y)$ *then true else false*), where f, g and $\circ$ are defined on the base and constructed types.

The LMW model also admits multiple values per cell. In this paper, we will not consider this extension. However, we will allow the array to consist of another complex type. For example, if $Typecolor = red, green, blue$ is a defined data type and A is a 2D array of type color, then $A[i,j].red$ denotes the red value of the (i,j)th cell, and $A.red$ is a shorthand to construct an array only with the red component of all cells projected out. Given an array A, we will also refer to the expression that generates that array as e_A. The elements of teh arrays will then be generated by the function $e_A[i] = \lambda e_A.i$.

In this paper, we will extend this model by creating the models and operators for a new type constructor called index-constrained set on multi-dimensional arrays to model multi-band wavelet features, and by defining regions with arbitrary shapes on single and constrained array-sets. Although we will follow closely the LMW model, all the examples will be in rather standard ML [?], augmented with array operations written in a rather straightforward notation. For the sake of clarity, we will avoid the complete (and rather complex) notation of [?] which the reader is invited to consult anyway. The completion of the examples and their casting in the complete notation should be straightforward.

We would like to reiterate that the primary rationale for adopting a complex object model for multimedia features is that we believe the current approach treating features as black boxes, severely restricts the expressiveness of retrieval operations that could be applied on features, and exposing the internal structures of features through a well- defined set of types and operations alleviates this limitation. At the same time, we do not profess the use of a general-purpose programming language to perform arbitrary operations on features, because that approach reduces the applicability of efficient search structures on large sets of data.

3 Regions in Arrays

Dealing with images often requires defining regions with arbitrary shapes. For this reason, we extend the array algebra with the concepts of *region* and *shape*. In order to do this, we first introduce the "universal null value" ϕ. The value ϕ doesn't have a type, but variables of any type can take the value ϕ. Moreover, if $\circ$ is an operator $\circ : T \times U \rightarrow V$ (that is, an operator that takes values of types T and U and returns a result of type V), we assume that ϕ is identified with the neutral element of $\circ$.

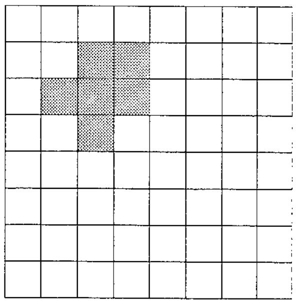


Figure 1:

Definition 3.1. *Given an expression e, a* region *S over A is a pair $S(e) = (A, E)$, where $A = [\![e | i_1 < e_1, \ldots, i_k < e_k]\!]$, and $E = \{u_1, \ldots, u_n\}$, $u_i \in \mathbb{N}^k$ is a collection of indices. The values $e_1 \cdots e_k$ are the size of the shape.*

Functionally, the region $S(e)$ is the array defined as

$$\lambda(\textit{if } i \in E \textit{ then } \lambda e.i \textit{ else } \phi).i \tag{1}$$

The region of Figure ??, for instance, is defined as $A = [\![e | 0 \leq i_1 \leq 7, 0 \leq i_2 \leq 7]\!]$,

$$E = \{(1,2),(1,3),(2,1),(2,2),(2,3),(3,2)\}$$

The dimensions of the array A are the intrinsic dimensions of the shape and, when the shape is applied to an array expression, they will be the dimensions of the resulting array. A region can also be applied to an array. In this case, the effect will be the same of applying the shape to the expression that generates the array, but the dimensions of the resulting array will be those of the array to which the shape is applied. For instance, given a two-dimensional array A of size 256×256, the expression $S(e_A)$ is a 9×9 array (the size of the shape), while the expression $S(A)$ is a 256×256 array containing the same shape and padded with ϕ values.

Formally, for a shape expression is still possible to define the function dim_k which, in this case, will return the dimension of the intrinsic size of the feature. Note that this function depends on the shape only, and is different from the same function applied to an array to which the shape function has been applied. For the shape above, for instance, it is $dim_2(S) = [9,9]$, $dim_2(S(e_A)) = [9,9]$, and $dim_2(A) = [256, 256]$.

Unions and intersections of regions can be defined in the standard way.

Region can betransformed into "floating" *shapes* by adding a parameter that represents their position in an array. If $j \in \mathbb{N}^k$, then a shape is a function $F(e) : \mathbb{N}^k \rightarrow [\![\tau]\!]$ that, for every instantiation of the location parameter j, generates a region with the upper-right hand corner located in position j. In other words, $F(e)(j)$ is the pair $F(e)(j) = (A, E)$ with A and E defined as before but where e generates the function

$$\lambda(\text{if } i \in E + j \text{ then } \lambda e.i - j \text{ else } \phi).i, \tag{2}$$

where $E + j = \{i + j : i \in E\}$. Note that, for correctness of notation, the shape is defined as a curried function, although in general the more common notation $F(e, j)$ will be used.

The usage of shapes outlined in this definition is analogous to the use of "masks" or "sliding masks" in image processing. Nevertheless, formally, a shape is not an array, but a mapping from array-defining expressions to arrays. A shape applied to an array-generating expression, like $s(e)$, is an array. A floating mask $F(e)$, on the other hand, is a function from indices to arrays, that is: $F(e) : \mathbb{N}^k \rightarrow [\![\tau]\!]$.

An operation often required when manipulating features is to determine whether a given property is verified for a given shape in any position of the array. This can be done using the `sweep` function, whose semantics can be defined for two dimensional arrays using monoid comprehension [?] as

$$\exists(F(e), A, P) = or\{P(F(e)(j), A) | j \leftarrow [0,0] \ldots [dim_1(A), dim_2(A)]\} \tag{3}$$

The definition can be easily extended to n dimensional arrays. Other operations can be defined in a similar way simply changing the target monoid. For instance, a selection operation, which returns all the locations in an array where the condition P is met for a shape can be defined as

$$\sigma(F(e), A, P) = set\{j | j \leftarrow [0,0] \ldots [dim_1(A), dim_2(A)], P(F(e)(j), A)\} \tag{4}$$

and the operation that counts how many times the condition P is satisfied is defined as

$$\Sigma(F(e), A, P) = +\{j | j \leftarrow [0,0] \ldots [dim_1(A), dim_2(A)], P(F(e)(j), A)\} \tag{5}$$

Shapes allows one to start expressing queries involving complex regions of the image.

Example 1. Find all arrays that agree with array A on the (localized) shape S.

```
fun within t,a,b =>
  let
    fn norm a => summapp(fn a, i => a[i]^2);
  in
    if norm a-b > t then false else true
query close_in_region a, s =>
  { d | d in ArrayDb.array1 and within t s(d) s(a) }
```

In this query, "ArrayDb" is the database, and "ArrayDb.array1" isolates a column that has been declared of array type.

Example 2. Find all arrays of size (256,256) where shape pattern $S(e_A)$ (A being a predefined array) occurs at least twice within the region (128,128), (255,255).

```
query shape_occur S t =>
  let
    fun P d t F =>
      within (t, F, d);
  in
  { d |
    dim(d, 1) = 256 and dim(d, 2) = 256 and
    count(128, 128, d, S, P(d, t)) >= 2 }
```

4 From Arrays to Index Constrained Arrays

Structurally, many wavelet transforms can be described as sets of arrays in which certain relations exists between elements of different arrays, namely, there are elements of different arrays that represent the same "physical location" in the image.

In order to arrive to such definition, we begin with the concept of *index constrained arrays.*

Definition 4.1. *An array B index-constrains an array B through the function f, written $A \xRightarrow{f} B$ if $A : [\![T]\!]_k$, $B : [\![M]\!]_k$, and $f : \mathbb{N}^k \to \mathbb{N}^k$ are such that*

1. *For all i, $dim_{k,i}(A) \le dim_{k,i}(B)$*
2. *The function f is defined for all legal values of indices of A, and for all such indices i, $f(i)$ is a legal index for B.*

A pair of arrays with the same size along all the dimensions are called iso-dimensional. The trivial index constrained pair for two iso-dimensional arrays A and B is $A \xRightarrow{\iota} B$, where ι is the identity transformation $\iota : \mathbb{N}^k \to \mathbb{N}^k$.

The inverse map f^{-1} maps elements of A into subsets of elements of B. The set of elements $f^{-1}(i)$, $i \in \mathbb{N}^k$ is called the *constraining set* for the element $A[i]$. Similarly, given a set of indices of A, $Q = \{i_1, \ldots, i_m\}$, the set $f^{-1}(Q)$ is the constraining set of the set $\{A[i_1], \ldots, A[i_m]\}$.

Note that the presence of the function f and its invertibility (in the set-of-indices sense illustrated aove) places some constraints on the sizes of the arrays. For instance, if A and B are two-dimensional arrays and $A \xRightarrow{f} B$ is an index constrained pair with $f(i,j) = (\lfloor i/2 \rfloor, \lfloor j/2 \rfloor)$, then if A has size $n \times n$, B must have size $2n \times 2n$. In this sense, the presence of the function f represents a structural constraint between pairs of arrays.

The definition of index constraint pair can be extended to sets of functions. Consider a set $\mathfrak{F}$ of functions $\mathbb{N}^k \to \mathbb{N}^k$ with the structure of a semi-group, that is $\iota \in \mathfrak{F}$ and $f_1, f_2 \in \mathfrak{F} \Leftarrow f_1 \circ f_2 \in \mathfrak{F}$. A pair of arrays is index constrained by the set $\mathfrak{F}$, written $A \xRightarrow{\mathfrak{F}} B$ if they are index constrained by any of the functions in $\mathfrak{F}$, that is,

$$A \xRightarrow{\mathfrak{F}} B \Rightarrow \exists f \in \mathfrak{F} : A \xRightarrow{f} B \tag{6}$$

Constrainment by a semi-group $\mathfrak{F}$ is reflexive and transitive. It is reflexive since for every array A it is trivially $A \xRightarrow{\iota} A$ (and, since $\mathfrak{F}$ is a semi-group, $\iota \in \mathfrak{F}$), and reflexive since $A \xRightarrow{f_1} B$ and $B \xRightarrow{f_2} C$ implies $A \xRightarrow{f_2 \circ f_1} C$.

Consider now a set of arrays $P = \{A^1, \ldots, A^N\}$ such that there are pairs of arrays A^u, A^v in P such that $A^u \xRightarrow{f} A^v$ for a function f belonging to a predefined semigroup $\mathfrak{F}$. Such a set is called an *index constrained set of arrays*, and will be denoted as $\langle P, \mathfrak{F} \rangle$.

The set of functions $\mathfrak{F}$ induces a structure in the set P. An array $A \in P$ is *initial* if there is no array $B \in P$ such that $B \xRightarrow{f} A$ for some $f \in \mathfrak{F}$. An array $A \in P$ is *terminal* if there is no array $B \in P$ such that $A \xRightarrow{f} B$ for some $f \in \mathfrak{F}$.

Definition 4.2. *An index-constrained set of arrays P is* linear *if*

1. *There is one and only one terminal array $A \in P$,*

2. *For any array $A \in P$, if $A \xRightarrow{f} A$ then f is the identity.*

The second point in the definition implies that, with the exception of the trivial correspondence between iso-dimensional arrays, there are no "loops" in P that is, the structure induced in P by the set $\mathfrak{F}$ (again, discarding the trivial constraints) is a tree.

Given an index-constrained set $C = \langle P, \mathfrak{F} \rangle$, two arrays A and B, and and index j for A, the function $\text{map}(A, B, j)$ returns the set of indices in the array B that are constrained to the index j of A. In other words:

$$\begin{aligned} \text{map}(A,B,f,j) \quad = \quad & \mathit{set}\,\{i | j \leftarrow [0,0], \ldots, [\mathit{dim}_1(A), \mathit{dim}_2(A)], \\ & i \leftarrow [0,0], \ldots, [\mathit{dim}_1(B), \mathit{dim}_2(B)], \\ & j = f(i) \vee i = f(j)\} \end{aligned} \tag{7}$$

Similarly, the function vmap returns an array built with the values indexed by the indices returned by map:

$$\text{vmap}(A,B,f,j) = \mathit{array}\{B[i] | i \in \text{map}(A,B,f,j)\} \tag{8}$$

The definition can be generalized so that, instead of a single index j one consider a whole region in the array A: the function $map(C, B, f, R(A))$, $f \in \mathfrak{F}$ returns the region on B induced by the function f applied to the shape $S(A)$:

$$\text{map}(C,B,f,R(A)) = \bigcup_{j \in R(A)} map(A,B,f,j) \tag{9}$$

Given a mapping f from A to B, in order to determine the ith element of the array $S'(B)$, one looks at all the elements of $S(A)$ that map to that element: if at least half of them are not null, then the value is $B[i]$, otherwise it is ϕ. If it is not the case that $A \xRightarrow{f} B$, then $S'(B)$ is the null array. Similarly, it is possible define the function $map(C, f^{-1}, B, S(A))$

In addition to the mapping function, several utility functions are defined. Given a list of arrays, the function `toList` transforms it into a list of sets of iso-dimensional arrays in such a way that the largest arrays come in first position, the second largest in second position, and so on. The function `first` extracts the set of the largest arrays, while the function `rest` extracts the rest of the list.

Also, given a set of arrays, the function `maxSize` extracts the subset of arrays with maximal size, and the function `minSize` extracts the set of arrays with minimal size.

As an example of applications, we will consider the definition of a multiresolution shape. Consider first a function `flt` that applies a shape expression to all arrays of a set:

```
fun flt( $P$, $S$) =
  if $|P|$ = 0
    null
  else if $|P|$ = 1
    $S(P)$
  else
    let $A = $ randomElement($P$);
    in
      $S(A) \cup$ flt($P - \{A\}$)
```

The function `splice` is then defined as:

```
fun splice($P, f, S$) =
  if $P = []$
    []
  else
  flt(first(P, S)) ++ splice(rest($P$), $f$, $S \circ f$);
```

5 Wavelet Features

We assume the readers to have a basic knowledge of wavelet functions and their properties [?]. Specifically, we develop on the idea that a wavelet function transforms an original signal into multiple frequency bands, including a low-pass band. Often, in an iterative process, the low-pass band is transformed again into similar frequency bands. Sometimes, the range of the signal of the ith iteration is reduced by half at the $(i+1)$th iteration. In this section we describe how wavelet features computed on multidimensional signals is modeled using the framework of index- constrained arrays developed in the previous section.

Let us define a labeled array as a tuple $\{\lambda, A\}$ such that the label λ is assigned from a domain Λ and A is an array. Let A_0 be the base array representing the original signal for which the wavelet transform is computed. A wavelet transform $W(A_0)$ is the linear index-constrained set of iso-dimensional sets of labeled arrays denoted as

$$W(A_0) = \langle [A_0, \{(\lambda_1^0, A_1^0), (\lambda_1^1, A_1^1), \ldots\}, \{(\lambda_2^0, A_2^0), (\lambda_2^1, A_2^1), \ldots\} \ldots], f \rangle \tag{10}$$

Each linear index-constrained set $\{(\lambda_j^0, A_j^0), (\lambda_j^1, A_j^1), \ldots\}$ is an iso-dimensional set that represents the frequency bands computed at any given resolution. The label of an array serves to identify the position of the array within the resolution level. If W is a Haar wavelet, then the label belongs to the domain $\{LH, HL, HH, LL\}$, on the other hand if W is a Gabor wavelet then its domain is a discrete set of angles ranging between 0 and 360. The index mapping function f is usually a transform that reduces an array of the previous resolution by half in each of its k dimensions, that is, for two dimensional arrays $f(i,j) = (\lfloor i/2, j/2 \rfloor)$. Additional constraints may be defined on the set depending on the nature of the wavelet transform. For example, for Gabor wavelets, an array at the i-th level can constrain the index of an array at the $(i+1)$-th level only if their labels are the same.

The operations defined on wavelet features are the same as those defined on index-constrained array, with the addition of the label selection operator σ_P^λ which, given a wavelet returns an index-constrained array containing all the elements of the wavelet that satisfy the condition P.

The previously defined operations on index-constrained set of array, in addition with the usual set, list, and selection operations, allow us to express complex queries involving wavelets.

Example 1. Find all images that agree with image I (represented as a wavelet transform) in the region S at a coarse resolution.

```
fun b $W, l, i$ =>
  if $i = l$
    W
  else
    b(rest($W$), $l$, $i+1$)
fun isolate $S, l, W$ =>
  splice(b($W, l, 0$), $W.f$, $S$);

query close_in_region W, l, s, t =>
  { I | I in ImageDb.wavelet and
    within ($t$, isolate($S, l, W$), isolate($S, l, I$) }
```

The function b takes the coarser resolution bands of the wavelet, while the function isolate projects the region S onto these bands. The functions splice and within are defined in the previous section.

Example 2. Find an image that agrees with image W in the shape $S1(i_0, j_0)$ and in the shape $S2(i_0, j_0)$, independently of the mutual positioning of the shapes. (This region based query is an extension of similar queries defined in [?]; the query can be expanded by defining functions that compare the relative positions of pairs of regions so that it can be imposed that the two regions be in the same spatial relation.)

```
fun cmp1 ($W, I, F, i, j, t$) =>
  let
    fun P $t, W, I$ => within($t, W, I$)
  in
  sweep($0, 0, I, S, P(S(W)(i, j))$);
query two_regions $W, F_0, i_0, j_0, F_1, i_1, j_1$ =>
  { I | I in ImageDb.wavelet and
    cmp1 ($W, I, F_0, i_0, j_0$) and cmp1 ($W, I, F_1, i_1, j_1$) }
```

6 Conclusions

In this paper we have proposed a data model for the representation, manipulation and query of wavelet features in multimedia databases. We argued that the representation of features as complex data types and the possibility to manipulate their structure is important for the design of powerful and efficient multimedia databases capable of transcending the limitations of the query-by-example model.

The model we presented is based on an extension of the array data type to include sets of arrays with certain structural relations. WE have showed that it is possible define wavelets as data types within such a framework and that a minimal set of operations (namely the array operations augmented with the map function) can be used to express complex queries involving wavelets.

The formalization of the structure of features and the definition of an algebra for their manipulation has two orders of advantages. First, the algebra is based on a limited number of operations on sets of labelled index-constrained arrays. The index-constrained arrays can be indexed using techniques that allow an efficient computation of these operation for large amounts of data, therefore making the computation of the individual operations efficient. Preliminary work carried out by us has indicated that efficient implementation of array selection operations can be obtained both by specialized indices and by special array representation in a relational database.

Second, by exposing the internal structure and the semantics of the operations, we create numerous possibilities for query optimization. Techniques for program normalization and query optimization based on the monoid comprehension calculus have been studied for quite some time [?], and our wavelet feature definition fits nicely in this framework.

References

[1] Ingrid Daubechies. *Ten lectures on wavelets.* Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[2] Leonidas Fegaras and David Maier. Towards an effective calculus for object query languages. In *Proceedings of SIGMOD '95, San Jose, CA*, pages 47–58, 1995.

[3] 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 and video content: the QBIC system. *IEEE Computer*, 1995.

[4] Amarnath Gupta and Simone Santini. Towards feature algebras in visual databases: The case for a histogram algebra. In *Proceedings of the IFIP Working Conference on Visual Databases (VDB5), Fukuoka (Japan)*, May 2000.

[5] Charles E. Jacobs, Adam Finkelstein, and Savid H. Salesin. Fast multiresolution image querying. In *Proceedings of SIGGRAPH 95, Los Angeles, CA.* ACM SIGGRAPH, New York, 1995.

[6] Leonid Libkin, Rona Machlin, and Limsoon Wong. A query language for multidimensional arrays: design, implementation and optimization techniques. In *Proceedings of SIGMOD '96, Montreal, Canada*, pages 228–239, 1996.

[7] Apostol Natsev, Rajeev Rastogu, and Kyuseok Shim. WARLUS: A similarity retrieval algorithm for image databases. In *Proceedings of SIGMOD '99, Philadelphia, PA*, pages 395–406, 1999.

[8] Arnold Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Image databases at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(1), 2001.

[9] Jeffrey Ullman. *Elements of ML Programming.* Prentice Hall, 1994.

Query Optimization in the Presence of Top-k Predicates

Lakshmi Priya Mahalingam and K. Selçuk Candan
Computer Science and Engineering Department,
Arizona State University
Email: {priyam,candan}@asu.edu

Abstract

In this paper, we present novel techniques for performing query optimization in databases, such as multimedia and web databases, which rely on top-k predicates. We propose an optimization model that (1) takes into account different binding patterns associated with query predicates and (2) considers the variations in the query result size (or coverage), depending on the execution order. We address the additional complexity and the well-known NP-complete nature of the query optimization problem by *adaptively* reducing the granularity of the search space. For this purpose, unlike the data histograms which capture the data distribution, we propose opt-histograms that capture the distribution of sub-query-plan values over many query optimization tasks.

1 Introduction

Multimedia database queries have different characteristics from queries in traditional databases. For instance, in traditional, such as relational, databases, it is usually the case that the number of results of a query is independent of the query execution order, but the execution cost depends on this order. In multimedia and web databases, however, due to the use of thresholds and top-k predicates and sub-queries [5, 11] the number of query results can also change depending on the query execution order. Due to these inherent differences between traditional databases and multimedia and web databases, we need novel query processing mechanisms. In particular, we need novel query optimization algorithms as the *top-k* natures of the query predicates render existing query optimization algorithms dysfunctional.

1.1 Motivating Example

Let us consider an SQL-like multimedia query that retrieves surveillance images that contain an alert pattern:

```
select location, time, image
from images
where surveillance_scans(location, time, image) and
      location = "entrance_gate" and
      extract_pattern(image, pattern) and
      match_pattern(pattern,"alert.gif").
```

In this application, surveillance images are continuously fed from the cameras into a database and the above query is executed repeatedly to identify the alert condition. Therefore, due to the nature of the application, it is essential that the query is evaluated as quickly as possible. Furthermore, since pattern extraction and matching are fuzzy operations, it is essential that the results of the query have associated *confidence levels* that can be used for deciding when to trigger the alert condition. Note that, in this application, we would like to identify the *best alert candidates* in the *shortest amount of time.*

While processing a query, we need to consider the following three issues characteristic to multimedia databases. Note that some of these, such as overloaded implementation of query predicates, are also seen in other database applications. However, altogether, the following three issues define the characteristics of multimedia databases:

Overloaded implementations of query predicates. Media-related predicates can be implemented by various user-defined functions or indexes corresponding to different ways the predicate can be invoked. For instance, the user defined predicate `extract_pattern(image,pattern)`, can have three different overloaded implementations:

- given an image, one implementation extracts a predetermined number of patterns using a pattern recognition function,
- given a pair of an image and a pattern, another one checks for the presence of the pattern in the image using a neural-net engine, and

- given a pattern, the last implementation retrieves all matching images using a cache of pre-extracted of pattern/image pairs maintained in the form of a multi-dimensional index structure that maps all known patterns into a feature space for quick access.

Each of these implementations can have drastically different behaviors in terms of the number of results returned. These are discussed next.

Thresholds and "top-k" predicates. The solution space in most media-related predicates is very large. For example, in the above example, every given pattern matches every other one to some degree. Therefore, providing all answers to a query is neither feasible nor desirable. Consequently, multimedia functions are usually implemented as *thresholding* or *top-k retrieval* functions. This means that a sub-query or a user defined function returns only a small, but most relevant, portion of all possible results. However, if two overloaded implementations of the same predicate define *relevance* differently, then they can return different portions of the result space.

This may cause certain complications in query optimization. One complication is that a query can result in different numbers of tuples depending on the execution plan chosen by the query optimizer. For example, let us assume that we can execute the multimedia query described earlier in two different ways:

- Find all surveillance images scanned at the "entrance gate"; find the patterns in each image; and return those images that contain a pattern which match "alert.gif".
- Find all patterns that resemble "alert.gif" using an index structure; find all images which contain one of these patterns; and return those images that are scanned at the "entrance gate".

In the last surveillance cycle, let us assume that the cameras scanned and fed into the system 100 images, 75 of which are coming from the camera at the "entrance gate". The image processing engine extracts 10 patterns per image. Due to the overlaps in the surveillance images, the number of unique patterns in all these images is 500, instead of 750 (75×10):

- Using the first execution plan, **(a1)** 100 images are retrieved and out of them 75 are chosen. **(a2)** Then the patterns present in these images are retrieved. If for this purpose, we are using a predicate which performs a top-5 retrieval, then 5 top patterns of each image are extracted, and **(a3)** the resulting 375 patterns are checked to see if they match the given pattern.
- When the second query execution plan is used, however, **(b1)** all patterns that match "alert.gif" (say 50 of them) are returned, **(b2)** all images which contain any one of these patterns are returned using an index structure, and **(b3)** out of these images, those that are located at the "entrance gate" are returned.

Note that there is no reason to expect that these two executions will return the same number of tuples. This difference is due to the fact that, at the second step of the process, different overloaded implementations (or binding patterns) of the predicate `extract_pattern(Image, Pattern)` use different retrieval policies: in **(a2)** the result space is pruned to the top-5 results, whereas in **(b2)** all results are maintained.

The necessity of new optimization techniques was noted in [2], where authors provide a novel similarity-based algebra (with new selection and join operators) and develop transformation that can be used in optimizing a query tree. We, on the other hand, use a query model where both selections and joins are modeled as predicates and, consequently, are able to state the query optimization problem as a join-ordering problem. We have studied semantics of various fuzzy conjunction, disjunction, and negation operators in [1]. Note that once an appropriate join order, which obeys the binding patterns of the predicates, is found some of the predicates will act as selections and others will act as join operators; implicitly achieving a rewritten query.

2 Optimization in the Presence of Top-k Predicates

Multimedia queries are generally processed in two ways: **(1)** using regular query processing techniques within database management systems and **(2)** using special merge techniques that benefit from the fact that results returned by most multimedia user-defined functions are ranked based on their scores [3, 6, 10, 11, 14]. In this paper, we develop a model which can lend itself to query optimization in the traditional sense instead of applying a merge strategy.

An optimization model assigns a value (to be minimized) to all partial or complete plans in the search space. It also determines the output size of the data stream for every operator and predicate in the plan. This plan must respect the available binding patterns of each query predicate. A binding pattern of a predicate specifies which attributes of a relation must be bound to access it. For example, a predicate $r(x, y)$, with two parameters can have four binding patterns:

- ff (or, alternatively, $\emptyset$) when both parameters are free,
- bf (or, alternatively, $\{x\}$) when x is bound and y is free,

- fb (or, alternatively, $\{y\}$) when x is free and y is bound, and
- bb (or, alternatively, $\{x, y\}$) when both parameters are bound.

Each binding pattern abstracts an overloaded implementation of a predicate in terms of an external function or an index structure. Therefore, given a multimedia query predicate, each binding pattern of predicate has different *cost*, and *size* parameters associated with it. This research aims at finding an optimal query execution plan, given a set of predicates, and the associated binding patterns, each with different execution cost and size characteristics. Therefore, we need an *optimization* model which captures not only the execution cost, but also the result size.

2.1 Query Execution Cost and Result Size (Fanout)

In this subsection, we introduce two parameters involved in the optimization model we propose, instead of the *table scan cost* and *selectivity* parameters traditionally used for optimizing queries:

- $cost$ and $fanout$.

These parameters have the following properties:

Property I: The $cost$ and $fanout$ parameters are defined *per binding pattern.* For example, given a predicate $r_1(x, y)$,

$$cost^{bf}(r_1(x, y))$$

denotes the expected cost of accessing r_1 by binding the parameter x to a constant and retrieving all matching values for y. Note that this definition is similar to the cost of a limited access pattern as used in [13].

Property II: The $cost$ parameter is defined as a function of the number of inputs to the bound parameters of the predicate. If a predicate, say $r_2(y, z)$ is executed after another one, say $r_1(x, y)$, during the processing of a join query

$$r_1(x, y) \wedge r_2(y, z),$$

then there will be a set, $\mathcal{Y}$, of y values that are already identified before r_2 is processed. Depending on the implementation of the query predicate, either each y value can be processed separately by r_2 to identify the corresponding z values or the entire set of y values can be passed into the corresponding media function at once to identify the corresponding $\langle y, z\rangle$ pairs. These two options would require different cost calculations, each of which is a function of the number of y values. For example, let us consider the two predicates introduced in the Introduction:

$$extract\_pattern(\text{“}image1.gif\text{''}, pattern) \wedge match\_pattern(pattern, \text{''}alert.gif\text{''}).$$

This query can be processed, by finding all patterns in the given image and, then, either (1) by matching each pattern separately with “alert.gif” or (2) by using a search structure, which contains all extracted patterns, for identifying those patterns that are close to "alert.gif". Note that the two above conditions are sometimes observed in traditional databases as well. For instance, the two query execution options described above and the difference between the corresponding cost calculations is analogous to difference between nested-loop and hash-joins. The next, and most significant, property however (at least currently) is more common in multimedia and web applications and we are not aware of any attempt to address it within the realm of query optimization.

Property III: The $fanout$ parameter denotes the expected number of outputs as a function of the number of inputs to the bound parameters of the predicate. This definition is inherently different from the definition of *selectivities* used in query optimization literature [12, 15]. For example,

$$fanout^{bf}(r_1(x, y))$$

denotes the expected number of $\langle x, y\rangle$ pairs that the predicate returns per each bound value of x (consequently, the fanout parameter can be larger than 1.0). The reason why we have to define a new parameter called $fanout$, instead of using the standard concept of selectivities is that,

- a top-k retrieval predicate (which may either be a user defined function or a top-k retrieval subquery) will have a predetermined number of results (k) independent of the value of x or the size of the actual result space. For example the predicate `extract_pattern(inputimage, outputpattern)` described in the Introduction extracts at most 5 patterns per input image, independent of the complexity of the image.

- in general, since we are not referring to a table with a known number of tuples, the definition of selectivity does not apply. For example the predicate `extract_pattern(inputimage, outputpattern)` does not perform a selection on a table of $\langle image, pattern \rangle$ pairs. Instead, it extracts whatever patterns are available in a given image.

Note that, whenever they are already available, it is possible to map selectivities to fanouts. For example, if we are given a table with schema $t(x,y)$ which has $num(t(x,y))$ rows and selectivity $sel^{bf}(t(x,y))$, then we can calculate the corresponding fanout as

$$fanout^{bf}(t(x,y)) = sel^{bf}(t(x,y)) \times num(t(x,y)).$$

In other words, this table will return on average $sel^{bf}(t(x,y)) \times num(t(x,y))$ pairs of $\langle x,y \rangle$ per each bound value of x. The reverse of this, however, is not true; that is, given a fanout, it is not always possible to map it into a selectivity as we are not always referring to a table with a known number of rows.

2.2 Modeling the Desirability of a Query Plan

The above semantics of predicate cost and fanout necessitate a corresponding model to evaluate the *desirability* [1] of a query plan. In general, given

- a set of predicates $\mathcal{P} = p_1, \ldots p_n$,
- a set of attributes $\mathcal{A} = a_1, \ldots, a_m$,
- a join query
  $$q(a_1, \ldots, a_m) = p_1(a_1, \ldots, a_m) \wedge \ldots \wedge p_n(a_1, \ldots, a_m),$$
- an input binding pattern, $in\_bound \subseteq \mathcal{A}$, which denotes the set of attributes that are bound in the beginning ($in\_free = \mathcal{A} - in\_bound$),
- an output binding pattern, $out\_bound \subseteq \mathcal{A}$, which denotes the set of attributes that are expected to be bound at the end of the process ($in\_bound \subset out\_bound$ and $out\_free = \mathcal{A} - out\_bound$), and
- for each predicate $p_i \in \mathcal{P}$, a binding function

  $bind_i : 2^{\mathcal{A}} \rightarrow \{2^{\mathcal{A}} \cup \perp\}$, such that given an input binding pattern, in, we have $bind_i(in) \supset in$ if the input binding pattern is allowed for this predicate and $bind_i(in) = \perp$ if it is not

  which describes the input/output binding relationships acceptable for this predicate,

we can define the cost and the fanout of the query as

$$\begin{array}{l}
\langle\ cost^{out\_bound}(q)\ ,\ fanout^{out\_bound}(q)\ \rangle\ =\ \textbf{best\_of}\ \{\langle cost, fanout\rangle\quad \textbf{such that}\\
\qquad cost = join\_cost(\langle cost^{pat_1}(p_1), fanout^{pat_1}(p_1)\rangle, \ldots, \langle cost^{pat_n}(p_n), fanout^{pat_n}(p_n)\rangle)\ \wedge\\
\qquad fanout = join\_fanout(fanout^{pat_1}(p_1) \ldots, fanout^{pat_n}(p_n))\ \wedge\\
\\
\qquad (\mathcal{A} - pat_1) \cup (\mathcal{A} - pat_2) \cup \ldots \cup (\mathcal{A} - pat_n) = in\_free\ \wedge\\
\qquad bind_1(pat_1) \cup bind_2(pat_2) \cup \ldots \cup bind_n(pat_n) = out\_bound\\
\},
\end{array}$$

where $join\_cost$ is the combined cost of the join as a function of the predicate costs and fanouts and $join\_fanout$ is the combined fanout as a function of the input fanouts. The last two conditions guarantee that input/output binding constraints are observed. Note that, we do not specify how the $join\_cost$ and $join\_fanout$ are computed, as these depend on the semantics of the join operator. The following example describes one way to compute $join\_cost$ and $join\_fanout$ assuming a nested-loop join.

Example 2.1 Using a nested loop join operator, the best way of computing $q(x,y,z) = r_1(x,y) \bowtie r_2(y,z)$ can be found by selecting the *best* among the following three options:

$$\begin{array}{l}
\langle cost^{bbb}(q(x,y,z)), fanout^{bbb}(q(x,y,z))\rangle = \quad \textbf{best\_of}\,\{\\
\qquad \langle cost^{ff}(r_1(x,y)) + fanout^{ff}(r_1(x,y)) \times cost^{bf}(r_2(y,z)), fanout^{ff}(r_1(x,y)) \times fanout^{bf}(r_2(y,z))\rangle,\\
\qquad \langle cost^{ff}(r_2(y,z)) + fanout^{ff}(r_2(y,z)) \times cost^{fb}(r_1(x,y)), fanout^{ff}(r_2(y,z)) \times fanout^{fb}(r_1(x,y))\rangle,\\
\qquad \langle cost^{ff}(r_1(x,y)) + cost^{ff}(r_2(y,z)) + fanout^{ff}(r_1(x,y)) \times fanout^{ff}(r_2(y,z)), fanout^{ff}(r_1(x,y)) \times fanout^{ff}(r_2(y,z))\rangle\}.
\end{array}$$

[1] Here, we use the term desirability instead of optimality as, in the literature, optimality is generally used to mean minimal cost.

This formulation describes the three alternative ways to join the given predicates from an input binding pattern $\emptyset$ (all attributes are free) to an output binding pattern $\{x, y, z\}$ (all attributes are bound). It further associates an execution cost ($cost$) and result size ($fanout$) for each option:

- In the first case, x and y have been instantiated by calling r_1 with free variables, and then these values have been validated by accessing r_2 with y bound. The cost is the access cost of r_1 when its attributes are free, plus the total access cost of r_2 when the common attribute, y, is bound for different possible values obtained from r_1. *The size of the results is the product of the expected result sizes associated with the corresponding binding patterns.*
- The second case is similar to the first case, except that the orders of r_1 and r_2 are exchanged.
- In the third case, both relations are independently accessed using both attributes free, and the results have been joined together. □

The **best_of** function is used to choose the best query plan among all the alternatives. In traditional models, it is relatively straightforward to choose the **best** plan among the alternative query processing plans. In the above example, traditional cost models would recognize that there are different costs associated with different ways to join these two relations. However, all possible executions would result in the same number of resulting tuples. Therefore, irrespective of the plan chosen, the final size, $fanout(r_1(x,y) \bowtie r_2(y,z))$, for the above example would be the same and we could choose the *best* plan by identifying the plan with the smallest $cost$. The proposed model, on the other hand, differs from the traditional models in that it has to account for different result sizes for different execution orders, a characteristic of multimedia and web databases as discussed in the Introduction. Hence, we can not choose the *best* plan in such a straight forward manner.

2.3 Alternative Ways to Define the Desirability of a Query Plan

In general, **best_of** will be a function of the cost and the fanout. We see three main alternative ways to define this measure in multimedia and web database applications:

Alternative I: best_of = min_cost. When the goal is to process the query as quickly as possible, the minimum *overall cost* (independent of the number of resulting tuples), we can define **best_of** as **min_cost**. In this model, given a set of $\langle cost, fanout \rangle$ pairs, we would choose the pair with the smallest cost.

This is similar to the *cost-based* optimization commonly used in databases. One major difference (and disadvantage) of this formulation is that, the principle of optimality may not be preserved. In other words, optimal (in terms of cost) plans may not have optimal (in terms of cost) subplans. This is due to the fact that logically equivalent sub-plans may have different fanouts, and when the cost parameter is used to select among alternative subplans, we may not reach an optimal overall plan. Consequently, a **min_cost** based model can not be applied recursively to sub-query plans and we can not use any recursively structured algorithm, such as the popular dynamic programming which assumes that all alternative ways to execute a sub-plan will result in the same number of tuples, to optimize queries.

Alternative II: best_of = min_unit_cost. When we are aiming to generate as many results as possible with the smallest cost, we can define the **best_of** as **min_unit_cost**, where *unit_cost* is defined as

$$unit\_cost = \frac{cost}{fanout}.$$

When the number of results may change based on the way query is processed, *unit_cost* may be a better performance indicator. One desirable property of a *unit_cost*-based model is that, under certain conditions, it may preserve the principle of optimality. For instance, if the cost and fanout is defined as in Example 2.1, then the principle holds.

Note that the concept of *unit-cost* is, in a sense, similar to the concept of *rank* [7, 12] used for dealing with expensive predicates. A *rank* is associated with every predicate in a query and these predicate ranks are used as heuristics to improve query plans. In the model we present, however, *unit-costs* are associated with alternative sub-plans, not with predicates themselves. Furthermore, even when *unit-costs* are associated with sub-plans that consist of only one predicate, they are functions of the corresponding binding patterns. In contrast, each predicate has one and only one *rank* associated with it [7, 12].

Alternative III: best_of = min_fanout. When the user wants to find a plan which returns the minimum number of results, we can define **best_of** as the plan with the smallest fanout. Note that, **min_fanout** is especially important when there is an access penalty associated with results (as it is the case when transmission of large media objects from a remote repository to a user is required). The **min_fanout**-based model, under certain conditions, may also preserve the principle of optimality. For instance, if the fanout is defined in a multiplicative way as in Example 2.1, then the principle holds.

2.4 Query Optimization Algorithms based on Desirability

When the **best_of** measure is defined as **min_unit_cost**, since it shows a recursive structure, we can benefit from traditional query processing algorithms, such as dynamic programming.

When the **min_cost** is required, however, we can no longer benefit from these algorithms. In this case, one solution is to use the **min_unit_cost** as a heuristic to reduce the search space at every level of a dynamic programming algorithm [2]. In other words, given a dynamic-programming based optimization algorithm, at each level, we can **(1)** rank sub-plans based on their *unit-costs*, **(2)** prune-away sub-plans with large *unit-costs*, and **(3)** consider only those plans with small *unit-costs.* The amount of pruning can be controlled to achieve different levels of optimization speed and optimality. There are three main options:

- *No pruning:* This approach will result in the most optimal plan as it exhaustively searches the whole search space. However, the drawback is that the search space is large and execution times are high.
- *Prune to the maximum level:* This essentially means that every stage the model prunes all the sub-plans except the one with the lowest rank. Though the approach increases execution speed largely, the error introduced is potentially very high.
- *Prune to a DBA-defined level:* In this approach, for each sub-goal, the algorithm considers the best k ranked subplans for further joins and others are neglected. These k sub-plans are used for further cost evaluations in the subsequent stages.

Since, **min_cost** is an important measure even in multimedia databases, it is important that the proposed heuristic works well and efficiently. Therefore, in the experiments section, we will evaluate the effectiveness of this approach.

3 Opt-Histograms: Adaptive Reduction of Search Space Granularity

Optimization is an NP-complete problem. The problem is rendered more drastic by the fact that, as discussed in Section 2, when we optimize for cost, we can not benefit from dynamic-programming based algorithms. As a solution, in Section 2, we proposed a sub-plan-rank based pruning approach to reduce the search space largely. Still, when k is large, the savings achieved in search space and execution time may not be very significant. Therefore, in addition to pruning the search space as described in Section 2, we propose to reduce the granularity of the search space to further bring down the complexity of query execution.

The number of sub-plans to be considered can be significantly reduced by approximating the costs of the sub-plans and choosing only the best of them. The desired level of approximation needed is described as the granularity of the distribution. If the granularity is small, there potentially is a considerable reduction in optimization time, but this introduces an error in the optimization.

We divide the whole search space into buckets, whose sizes are determined by the granularity desired. All sub-plans that belong to a particular interval of values are placed in the same bucket. In each step of query-optimization, instead of using all sub-plans, only one sub-plan from every bucket is used. Due to this approximation, the search-space size is controlled.

Example 3.1 Let us have eight alternative sub-plans that we have to consider,each with the following associated costs:

Sub-plan 1 = 120,	Sub-plan 2 = 130,	Sub-plan 3 = 240,	Sub-plan 4 = 520,
Sub-plan 5 = 650,	Sub-plan 6 = 110,	Sub-plan 7 = 1000,	Sub-plan 8 = 620.

Let us also assume that we use a bucket width of 100 unit costs to reduce the granularity of the search space. Instead of maintaining all eight alternatives, we need to maintain only five alternatives:

Sub-plan 6 = 110,	Sub-plan 3 = 240,	Sub-plan 4 = 520,	Sub-plan 8 = 620,	Sub-plan 7 = 1000.

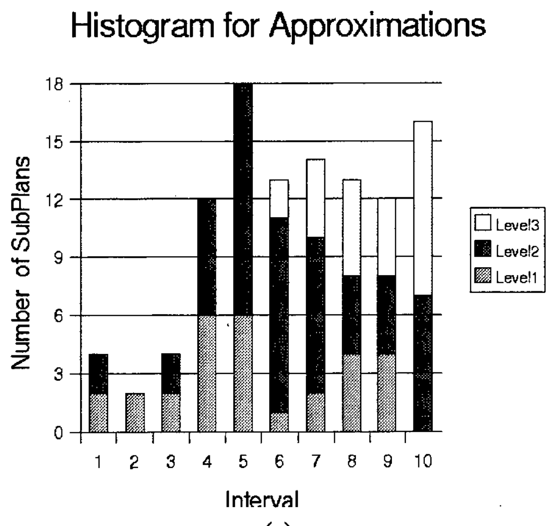
□

Note that in the above example, the buckets are placed uniformly across the search space starting from cost 0. However, in general, identifying the boundaries and widths of the buckets is not trivial. First of all, since the execution cost is likely to increase with the number of predicates in the query, at different levels of query optimization, different bucket boundaries may need to be used. Secondly, since the search space has two dimensions, *cost* and *size*, we need to identify different bucketing strategies for each dimension. In this section, we introduce *opt-histograms* that can be used for intelligently choosing the granularity (places of bucket boundaries) of the search space. Opt-histograms describe the optimization statistics accumulated by observing similar queries optimized over a period of time.

3.1 Opt-Histograms

An opt-histogram is generated by analyzing the query optimization history. Figure 1(a) shows an example opt-histogram generated using a set of similar queries. The opt-histogram shows the distribution of the estimated costs of sub-plans that belong to different

[2] Since dynamic programming-based query optimization algorithms are common and well-understood, we do not provide the pseudo-code of such an algorithm in this paper



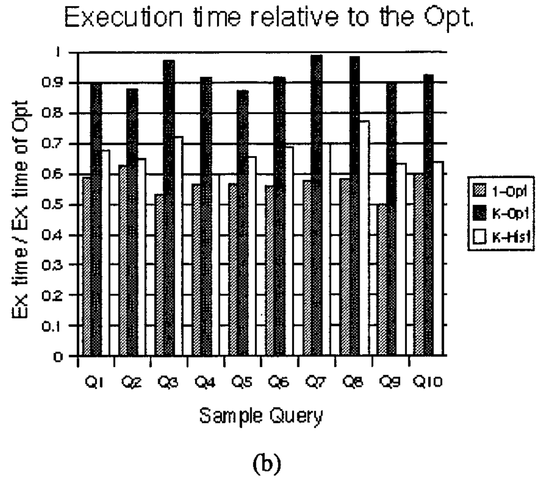


Figure 1: (a) An example opt-histogram, which shows that the relevant space of costs varies with the query optimization stages. (b) Average optimization time relative to the optimization time required by the optimal-cost algorithm

level of join ordering. This histogram was generated over 1000 optimization tasks using a 4-relation query. The levels in the figure denote the 2-, 3-, 4- way join stages of the query optimization process. As shown in Figure 1(a), we observed a consistent behavior of sub-plan execution costs in different query optimization stages. Note that, according to this figure, relevant regions of the cost-space shifts at different stages of the query optimization process. Therefore, different bucket boundaries should be used at different stages of query optimization.

Note that, one property of the opt-histograms that is special to the multimedia and web databases, is that sizes show different characteristics than the cost. Sizes can be defined multiplicatively (unlike the additively defined cost). Therefore, they require non-uniform granularity reductions of the space. In this paper, we do not discuss the details of the opt-histogram creation for size.

In Section 2, we mentioned that a dynamic-programming based algorithm can be implemented by selecting the best k ranked sub-plans for every two-way join, when we are dealing with the **best_cost** metric which does not preserve optimality for sub-plans. We can further reduce the search space, by coalescing the sub-plans using a low granularity search space. All plans that fall into a particular bucket are approximated as being equivalent and this reduces the search space further. The salient features of the described approach are that it

- approximates the sub-plans by reducing the granularity of search space,
- uses an optimization histogram to reduce the granularity of the space adaptively, and
- generates this histogram by observing the behavior of the algorithm, for similar queries, at different levels of join ordering.

Next, we describe experiment results to justify that highlight the issues raised in this paper and that evaluate the effectiveness of the proposed solutions.

4 Experimental Results

In order to evaluate the effectiveness of the techniques presented in this paper, we ran a set of experiments. In order to observe the behavior of the proposed query optimization algorithm under very different conditions, we opted to use synthetically generated data. For this purpose, we used a set of join-queries, each with different number of joins and joining attributes. In different runs, each predicate got associated with randomly generated data that follow random *cost* and *size* patterns. These patterns are generated keeping in mind the characteristics of multimedia and web predicates:

- *Cost of predicates:* The cost of predicates varied greatly. In fact, the cost of different binding patterns of individual predicates also varied. There are expensive binding patterns which correspond to user-defined functions which require media processing and there are cheaper binding patterns which correspond to use of appropriate index structures.

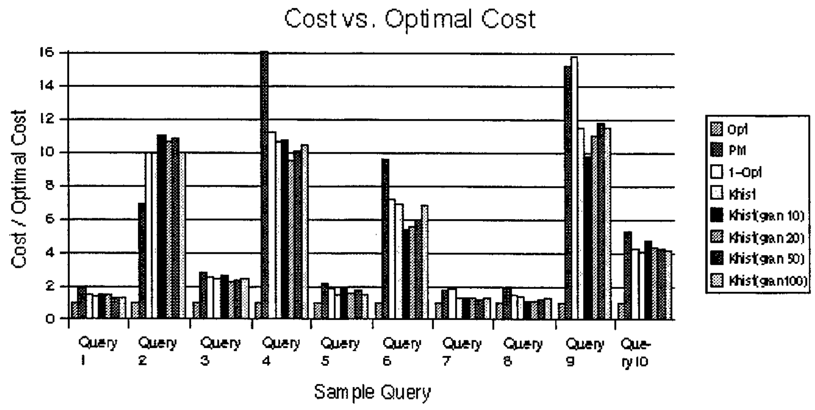


Figure 2: Relative optimality of various algorithms

- *Size:* The size factor of predicates and their binding patterns also varied to reflect the top-k retrieval functions which return a predetermined number of results for each input as well as thresholding functions which returns a portion of the database determined by the threshold value as well as the inherent distribution of the data.

‘ We have varied these expected cost and size values of each predicate across different runs to observe the performance of the proposed optimization algorithm under different input parameters. In order to compare the performances (in terms of time as well as the closeness to optimality) of various approaches, we have implemented the following algorithms:

- *Optimal:* This algorithm performs an exhaustive search. It enumerates and expands all possible plans with no pruning or approximations.
- *Predicate migration:* We have also implemented the predicate migration algorithm [12]. Note that predicate migration uses only one rank per predicate. It is essentially a heuristic when applied to the model presented in this paper. Therefore, to evaluate its performance conservatively, for each predicate we used the worst rank among all its binding patterns.
- *1-Optimal:* This algorithm prunes the search space choosing only one best plan for each sub-goal.
- *K-Optimal:* This algorithm keeps k best plans for each sub-goal (3 in these experiments).
- *K-Hist:* This algorithm approximates the top k ranked sub-plans using a reduced granularity search space.

The following sections discuss how these algorithms perform with respect to cost and size factors.

4.1 Complexity

Figure 1(b) shows execution cost of various algorithms for different queries. Since the optimal algorithm performs an exhaustive search, it requires the highest execution time. The predicate migration considers only one plan per level of join and hence the search space is very small. Therefore, it performs the fastest optimization (not shown in the figure). 1-opt also performs optimization quickly (in the same order of the predicate migration) when compared to others as it prunes the search space to the greatest level possible and thus time for enumerating the entire search space is reduced. K-opt searches a larger search space and hence performs slower than 1-opt and faster than optimal algorithms. The k-hist algorithm reduces the search space by reducing the granularity of the search space and approximating the results at every stage. Thus, a considerable saving in execution times is achieved.

4.2 Degree of Optimality

Figure 2 shows the relative behavior of different algorithms in terms of finding the best query execution plan. The experiments were performed for a small sample of queries with different predicate costs and sizes[3]. The x-axis shows a sample of queries and y-axis shows the ratio of the cost of each resulting plan to the cost of the cheapest plan (found by the optimal algorithm). According to this figure, 1-opt over-prunes the search space and the costs deviate from the optimal largely. The k-opt algorithm

[3]Since in this paper we do not concentrate on the question of whether different algorithms works better for different queries, we do not list the actual queries used in this example. But, it is important to note that the relative patterns is more or less the same across different queries.

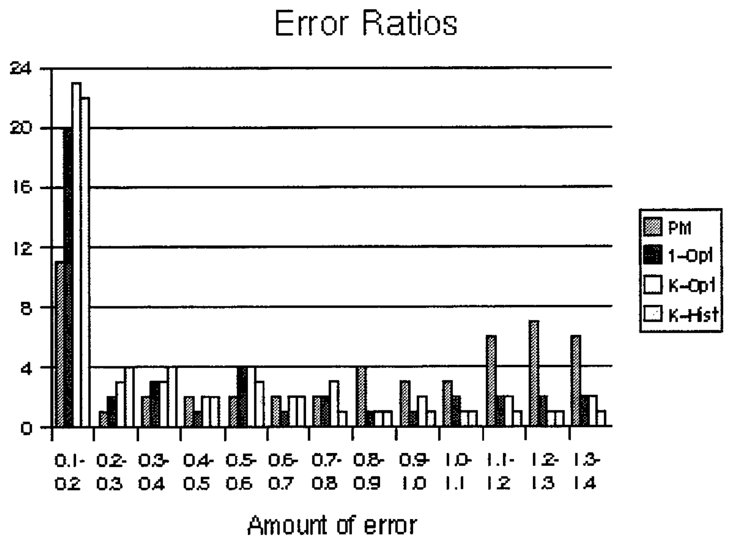


Figure 3: Error distribution (error boundaries are selected in 0.1 increments (i.e., 0.0-to-0.1, 0.1-to-0.2, etc.)

performs relatively well and k-hist (shown with different granularities) approximates the results of k-opt well. Note that, as the granularity increases, k-hist resembles the k-opt algorithm. Note that, since it is essentially a heuristic when applied to the novel model, predicate migration behaves less predictably.

Note that although it shows the relative behavior of the different optimization algorithms, this small sample does not represent the degree of optimality of these algorithms. A better way to observe the degree of optimality is to plot an histogram of their error relative to the optimal result:

$$Error = \frac{Cost - Optimal\_cost}{Optimal\_cost},$$

over a large number of queries. The experimental results are plotted in Figure 3. In this figure, each bucket on the x-axis corresponds to an error range of size 0.1 (that is intervals corresponds to the error ranges 0.0-0.1, 0.1-0.2 etc.) and the y-axis corresponds to the number queries which falls in each bucket (or error range). Note that according to this figure, k-opt algorithm has the highest number of correct estimations (error range within the 0.0-0.1 bucket). Note, on the other hand, that surprisingly, k-hist algorithm seems to cause a smaller number of large errors than the k-opt algorithm. Although, this is interesting, we do not believe that it is significant and generalizable.

5 Related Work

The cost model used by most researchers does not consider the effect of varying result sizes. Selinger *et al.* [15] find the cost of a join assuming a constant product of cardinalities independent of query execution order. Recently, there has been efforts realizing the fuzzy and probabilistic nature of new database applications (such as [2, 3, 10, 11, 14]) and benefit from this nature in top-k retrieval [5]. Most of the existing work in this area concentrate on identification of efficient query processing techniques for queries with fuzzy predicates. In this paper, we concentrate on optimizing queries; i.e., identifying efficient query plans when there are various alternatives. In [6], Chaudhuri and Gravano discuss cost-based query optimization techniques for such *filter* queries. In essence they convert top-k retrieval queries to thresholding (filtering) queries that can be processed using more traditional database processors. In [9] Donjerkovic and Ramakrishnan looked at the effect of top-k evaluations on the query optimization task. Their focus was to minimize the overall query execution cost by trying to minimize the effect of restarts due to too little answers. In [3], we presented an approximate query evaluation algorithm that builds on [10, 11] to address the existence of non-progressive fuzzy predicates. This algorithm uses the statistical properties of the minimizes the unnecessary accesses to non-progressive predicates, while providing error-bounds on the top-k retrieval results.

Hellerstein [12] presents various techniques for efficiently optimizing declarative queries that contain time consuming methods. The approach used for optimization in this work is construction of a plan tree as a first step with each sub-plan represented as a separate stream. Then each of the streams are sorted based on a rank[4]. Then a differential cost function prunes the results by pulling the top ranked predicates. Hellerstein discusses four main optimization strategies, namely pushdown with rank ordering, pull up costly sub-plans, pull the costly ranked predicates while performing a join based on join ranks rather than predicate ranks, and predicate migration, which considers the least cost plan at every stage of join. LDL [8] treats predicates as relations. One major drawback of the optimization approach used in [8] is over eager pull up decisions. The algorithm makes decisions looking at only the immediate sub-plans. Another drawback of this approach is that the complecity is exponential in terms of number of relations and number of expensive predicates. [4, 7] provide a comparison of LDL and predicate migration approaches. Their research focuses on pruning by ranks where $Rank = \frac{Cost}{(1-Selectivity)}$. Chaudhuri and Shim's algorithm [7] performs well with the assumption that predicates with smaller ranks are always applied earlier than others. The assumption however holds for restrictive predicates only (predicates where all the parameters are bound), which ceases to be the case when varying binding patterns are used. Another assumption of their work is that there is only one rank value per predicate which again may not always hold. The complexity is polynomial in the number of user-defined functions. The heuristic is very successful in pruning unwanted results. The push down heuristic pushes all expensive predicates down or defers their evaluation until the last join.

6 Conclusion

In this paper, we presented a new query optimization model for multimedia and web databases, which use top-k predicates. Unlike the previous work, we considered non-restrictive predicates in query optimization. This model takes into account different bindings for every predicate, and it recognizes the fact that a query may have different number of results (or coverages), depending on the query execution plan. We used an efficient pruning criteria that prunes, based on cost and size factors, in the sub-plan level. The proposed approach addresses the well-known NP-complete nature of the query optimization problem by *adaptively* reducing the granularity of the search space. For this purpose, unlike the data histograms which capture the data distribution, we use opt-histograms that capture the distribution of sub-query-plan values over many query optimization tasks.

Acknowledgements

We thank to Chris Mayer for his helpful comments.

References

[1] K.S. Candan and W.-S. Li. *On Similarity Measures for Multimedia Database Applications*, KAIS journal, 3(1), pp. 30-51, 2001.

[2] S. Adalı, P. Bonatti, M.L. Sapino, and V.S. Subrahmanian. *A Multi-Similarity Algebra.* SIGMOD 98.

[3] K.S. Candan, W.-S. Li, and M.L. Priya. *Similarity-based Ranking and Query Processing in Multimedia Databases.* DKE 35(3): 259-298, 2000.

[4] Surajit Chaudhuri, *An Overview of Query Optimization in Relational Systems*, Principles Of Database Systems, 1998.

[5] Surajit Chaudhuri and Luis Gravano, *Evaluating Top-k Selection Queries*, VLDB 1999, pp 397-410, 1999.

[6] Surajit Chaudhuri and Luis Gravano, *Optimizing Queries over Multimedia Repositories*, SIGMOD 1996, pp. 91-102.

[7] Surajit Chaudhuri and Kyuseok Shim, *Optimization of Queries with User-Defined Predicates,* VLDB 96, pp. 87-98.

[8] D. Chimenti, R. Gamboa, and R. Krishnamurthy. *Towards on Open Architecture for LDL,* VLDB 1989, pp. 195-203, 1989.

[9] Donko Donjerkovic and Raghu Ramakrishnan, *Probabilistic Optimization of Top n Queries.* VLDB, pp. 411-422. 1999.

[10] R. Fagin. *Combining Fuzzy Information from Multiple Systems.* 15th ACM Symposium on Principles of Database Systems, pp. 216-226, June 1996.

[11] Ronald Fagin. *Fuzzy Queries in Multimedia Database Systems*, Principles of Database Systems, Seattle, WA, 1998.

[12] Joseph M. Hellerstein, *Optimization Techniques for Queries with Expensive Methods,* Association of Computing Machinery, Transactions on Database Systems, Vol.23, No.2, June 1998, pp 113-157.

[13] Alon Levy, Ioana Manolesu, Dan Suciu, Daniela Florescu, *Query Optimization in the Presence of Limited Access Patterns*, SIGMOD 1999, pp. 311-322, 1999.

[14] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T.S. Huang, *Supporting Ranked Boolean Similarity Queries in MARS*, TKDE, (10)6, pp. 905-925, 1998.

[15] P. Griffiths Selinger, D.D. Chamberlin, R.A Lorie, T.G. Price, *Access Path Selection in a Relational Database Management System*, SIGMOD 1979, pp.23-34, 1979.

[4]The definition of rank of a predicate is $Rank = \frac{(Selectivity-1)}{Cost}$

Comparing Semi-Structured Documents via Graph Probing

Daniel Lopresti *Gordon Wilfong*

Bell Labs, Lucent Technologies Inc.
600 Mountain Avenue
Murray Hill, NJ 07974 USA
{dpl,gtw}@research.bell-labs.com

Abstract

In this paper, we describe our first steps towards adapting a new approach for graph comparison known as *graph probing* to allow for the pre-computation of a compact, efficient probe set for databases of graph-structured documents (*e.g.*, Web pages coded in HTML). We consider both the comparison of two graphs in their entirety, as well as determining whether one graph contains a subgraph that closely matches the other. After presenting an overview of work in progress, we provide some preliminary experimental results and suggest directions for future research.

1 Introduction

Graphs are a fundamental representation in much of computer science, including the analysis of both traditional and multimedia documents. Algorithms for higher-level document understanding tasks often use graphs to encode logical structure. HTML pages are usually regarded as tree-structured, while the WWW itself is an enormous, dynamic multigraph. Much work on attempting to extract information from Web pages makes explicit or implicit use of graph representations [3, 4, 7, 11, 13].

It follows, then, that the ability to compare two graphs is also basic functionality, as demonstrated by such diverse applications as pattern recognition, performance evaluation, query-by-structure, wrapper generation for information extraction, etc. Because most problems relating to graph comparison have no known efficient, guaranteed-optimal solution, researchers have developed a wide range of heuristics. For the problem of determining isomorphism, for example, many heuristics rely on the existence of certain *vertex invariants*, which consist of a value $f(v)$ assigned to each vertex v, so that under any isomorphism I, if $I(v) = v'$ then $f(v) = f(v')$. One commonly used vertex invariant is the degree of a vertex. It has been shown that for random graphs, there is a simple linear time test for checking if two graphs are isomorphic that is based on the degree of the vertices, and this test succeeds with high probability [1]. In fact **nauty**, a successful software package for determining graph isomorphism (see [10]), relies on such vertex invariants.

This observation can be seen as forming the basis for *graph probing*, a paradigm we have recently begun exploring for graph comparison [5, 6, 9]. However, we desire more than a simple "yes/no" answer; we are interested in quantifying the similarity between graphs, not just in whether they may be isomorphic. Conceptually, the idea is to place each of the two graphs under study inside a "black box" capable of evaluating a set of graph-oriented operations. We then pose a series of probes and correlate the responses of the two systems. This process is depicted in Figure 1.

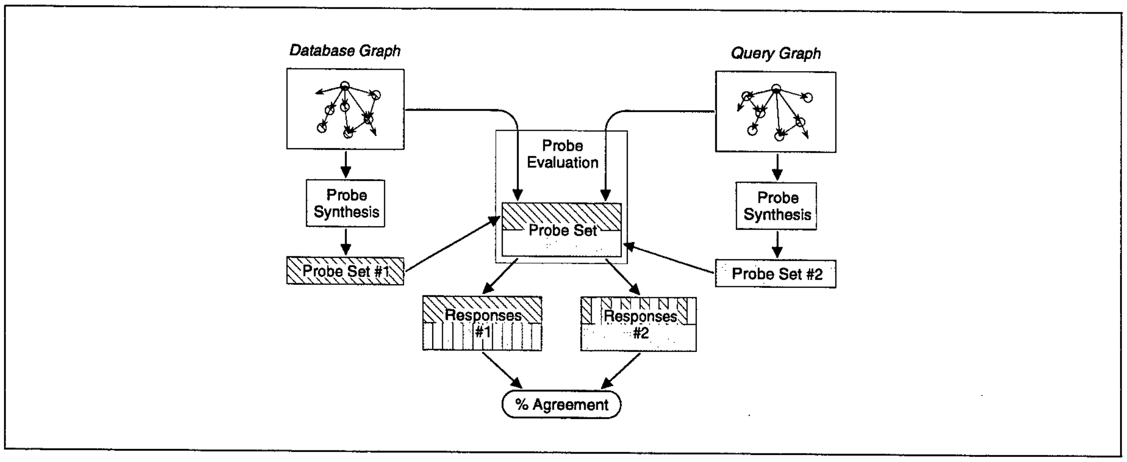


Figure 1: Overview of graph probing.

Our past work in the area treats graph probing as an on-line process; both the query graph and the database graph are available at run-time for synthesizing the probe set. While this is an appropriate assumption when one is comparing, say, the output of a recognition algorithm with its associated ground-truth, it is not a workable model for retrieval applications when the database contains anything other than a small number of documents.

In the present paper, we describe our first steps towards adapting the graph probing paradigm to allow for the pre-computation of a compact, efficient probe set for databases of graph-structured documents in general, and Web pages coded in HTML in particular. The comparison of two graphs in their entirety is considered, as well as determining whether one graph contains a subgraph that closely matches the other. We present an overview of work in progress, as well as some preliminary experimental results.

2 Related Work

Graph comparison is an important yet difficult problem, so it should come as no surprise that a large number of researchers have proposed heuristics or solutions designed for special cases. For example, Bunke and Messmer present a decision-tree-based precomputation scheme for solving the subgraph isomorphism problem [2], although their data structure can be exponential in the size of the database graphs in the worst case.

Lazarescu *et al.* propose a machine learning approach to building decision trees for eliminating from further consideration graphs that cannot possibly be isomorphic to a given query [8]. While they employ a similar set of features to the ones we use, they do not consider the approximate matching or subgraph problems.

Papadopoulos and Manolopoulos discuss an idea that is philosophically quite similar to ours [12]. However, they focus on a single invariant: vertex degree. It is clear this is not sufficient for catching all of the interesting differences that can arise between HTML documents. We would like to determine a range of possible graph features that can be used to distinguish the sorts of effects that arise in practice. Moreover, their histogram technique is applied only to the problem of comparing complete graphs, whereas we wish to examine the subgraph matching problem as well.

Valiente and Martínez describe an approach for subgraph pattern-matching based on finding homomorphic images of every connected component in the query [14]. Again, the worst-case time complexity is exponential, but such features could also perhaps be incorporated in the heuristics we are about to present.

Instead of trying to solve the problem for graphs in general, some leeway can be had be limiting the discussion to trees, for which efficient comparison algorithms are known. Schlieder and Naumann consider a problem closely related to ours: error-tolerant embedding of trees to judge the similarity of XML documents [13]. Likewise, Dubois *et al.* write about tree embedding for searching databases of semi-structured multimedia documents and for query-by-example [3].

The WebMARS system, as presented by Ortega-Binderberger *et al.* [11], models Web documents via their parse trees. Queries are likewise treated as trees, although they encode hierarchy for individual object types (*e.g.*, text, images) and do not represent the same sorts of inter-object relationships that the mark-up in Web documents encodes. Matching only takes place between the leaves of the query tree and all possible "chains" in the document tree (*i.e.*, paths leading in the direction from the root to a leaf). The match values are then propagated upwards towards the root of the query tree over edges that can be weighted to reflect the importance of that particular component. Hence, there is an asymmetry between queries and documents. In any case, the graph model we would like to support is more general than simple trees, allowing both cross-and back-edges.

3 A Formalism for Graph Probing

In this section we formalize the concept of graph probing as a way of quantifying graph similarity. Our goal is to relate probing, which is a heuristic, to more rigorous but harder-to-compute graph edit distance models.

While ultimately we are interested in a more general class of graphs, to begin with let $G_1^u = (V_1, E_1)$ and $G_2^u = (V_2, E_2)$ be two undirected graphs. Consider a graph editing model that allows the following basic operations: (1) delete an edge, (2) insert an edge, (3) delete an isolated vertex, (4) insert an isolated vertex. It should be clear that such operations can be used to edit any graph into any other graph. The minimum number of operations needed to edit G_1^u into G_2^u is the undirected graph edit distance, $dist^u(G_1^u, G_2^u)$. As noted previously, there is no known algorithm for efficiently computing this distance in general.

Now consider a probing procedure that, for a specific vertex degree n, asks the following question: "How many vertices with degree n are present in graph $G^u = (V, E)$?" Let PR_{1a} collect the responses for all vertex degrees represented in the graph (the response for all other vertex degrees is, of course, implicitly "0"),

$$PR_{1a}(G^u) \equiv [\, |\{v \in V \mid deg(v) = n\}| \;\text{ for } n \in \{deg(v) \mid v \in V\}\,] \qquad (1)$$

Then define $probe^u(G_1^u, G_2^u) \equiv PR_{1a}(G_1^u) - PR_{1a}(G_2^u)$ as the L_1 norm of the two vectors; that is, $probe^u$ is the magnitude of the difference between the two sets of probing results.

Theorem 1 *Under the undirected graph model and its associated edit model, $probe^u$ is, within a factor of four, a lower bound on the true edit distance between any two graphs, $probe^u(G_1^u, G_2^u) \leq 4 \cdot dist^u(G_1^u, G_2^u)$.*

The time needed to perform the above probing procedure is $O(\max(|V_1|, |E_1|, |V_2|, |E_2|))$. In the off-line case, we can precompute the probes and their responses for one of the graphs. The result is exactly the same as in the on-line case. While the worst-case time complexity remains unchanged, the precomputation and an efficient coding of its output can yield a substantial savings.

Unfortunately, there is no guarantee that the above bound is particularly tight. Indeed, it is not even as tight as the bound that can be derived for the measure described in [12], although it does appear to be easier to generalize, a point that will become important shortly. Still, as a lower bound it does provide a potentially useful filter, which is our goal.

To proceed to the case of directed graphs, we can consider the same set of editing operations (recognizing that the edges are now directed) and change the probes to be: "How many vertices with in-degree m and out-degree n are present in graph G^d?" Observations similar to those made above concerning lower bounds and computation time for undirected graphs apply here as well (including a theorem for directed graphs analogous to Theorem 1). It can also be shown that the bound returned by the new, more specific class of probes is at least as good as the original class and sometimes better: $probe^u(G_1^d, G_2^d) \leq probe^d(G_1^d, G_2^d)$.

Now generalize the graph model further so that vertices and edges are potentially labeled by a type. For example, vertices might be labeled as corresponding to HTML structure tags (*e.g.*, section heading, paragraph, table), while edges are labeled to represent relationships between structures (*e.g.*, contains, next, hypertext reference). To handle such attribute graphs, the edit model previously defined must be expanded to include additional operations: (5) change the type of an edge, (6) change the type of a vertex. The edges and vertices created through insertion operations can be assigned any type initially.

In terms of probing, note that now two graphs can be different (in terms of the types of their vertices and edges) and yet appear to be structurally identical. To deal with this, the probes for counting in- and out-degrees are made specific to edge type. Suppose there are α different edge labels $l_1, \ldots, l_\alpha$. The edge structure of a given vertex can then be represented as a 2α-tuple of non-negative integers, $(x_1, \ldots, x_\alpha, y_1, \ldots, y_\alpha)$, if the vertex has exactly x_i incoming edges labeled l_i and exactly y_j outgoing edges labeled l_j for $1 \leq i, j \leq \alpha$. Then a typical probe will have the form: "How many vertices with edge structure $(x_1, \ldots, x_\alpha, y_1, \ldots, y_\alpha)$ are present in graph G^a?" We also need to add a new class of probes focusing on just the vertices and their types: "How many vertices labeled *vtype* are present in graph G^a?" Let PR_{1c} collect the responses for vertex in- and out-degrees and their respective edge types, and let PR_2 collect the responses for vertex types. Define $probe^a(G_1^a, G_2^a) \equiv (PR_{1c}(G_1^a) - PR_{1c}(G_2^a)) + (PR_2(G_1^a) - PR_2(G_2^a))$. We then have:

Graph Model	*Edit Model*	*Probe Model*	*Bound*
Undirected	(1) delete edge (2) insert edge (3) delete vertex (4) insert vertex	(a) vertex degree	$probe^u(G_1^u, G_2^u) \leq 4 \cdot dist^u(G_1^u, G_2^u)$
Directed	as above	(a) vertex in- and out-degree	$probe^d(G_1^d, G_2^d) \leq 4 \cdot dist^d(G_1^d, G_2^d)$
Attribute	as above, plus (5) change edge type (6) change vertex type	(a) vertex in- and out-degree by edge type (b) vertex type	$probe^a(G_1^a, G_2^a) \leq 4 \cdot dist^a(G_1^a, G_2^a)$

Table 1: Summary of the various models.

Theorem 2 *Under the attribute graph model and its associated edit model, $probe^a$ is, within a factor of four, a lower bound on the true edit distance between any two graphs, $probe^a(G_1^a, G_2^a) \leq 4 \cdot dist^a(G_1^a, G_2^a)$.*

Moreover, $probe^u(G_1^a, G_2^a) \leq probe^d(G_1^a, G_2^a) \leq probe^a(G_1^a, G_2^a)$.

The precomputation needed for each graph is as follows. Computing the edge structures of all the vertices takes total time $O(|E| + \alpha|V|)$. These $|V|$ tuples can then be lexicographically sorted in $O(\alpha(d + |V|))$ time, where d is the maximum number of edges incident on any vertex. Then a simple pass through the sorted list allows us to compute the number of vertices in each of the (non-empty) classes in additional time $O(\alpha|V|)$. Thus the total precomputation time is $O(\alpha(d + |V|) + |E|)$. Since α and d are likely to be small constants, the time is essentially the same as for the case of undirected graphs.

Table 1 summarizes the results of this section.

4 Graph Probing for Semi-Structured Documents

The attribute graph model we assume for HTML documents includes the standard tree-structured hierarchy generated when parsing the tags (the "contains"/"contained-by" relationship). In addition, we also make use of the order in which content and the various substructures are encountered (in many cases this corresponds to the natural reading order for the material in question). We represent this via "next"/"previous" cross-edges that connect vertices at a given level in the hierarchy, rather than assuming an implicit fixed ordering on the children of a vertex as some other researchers have done. Lastly, we record hyperlinks as either back-edges (in the case of targets on the same page) or a distinguished vertex type (in the case of external references). No provision is currently made for incoming links from outside documents. See Figure 4 for an example from our small test database.

Recall that we have defined two probe classes for these kinds of graphs:

Class 1c These probes examine the vertex and edge structure of the graph by counting in- and out-degrees, tabulating different types of incoming and outgoing edges separately.

Class 2 These probes count the occurrences of a given type of vertex in the graph.

When comparing two graphs in their entirety, it suffices to correlate their responses to the probes using the L_1 norm as described earlier. For the problem of subgraph matching, however, we cannot expect to be able to compare directly the outputs for the larger graph to those for the smaller. For example, consider a query graph consisting of a single table that corresponds to a table in some database document, but where that document also has dozens of other, unrelated tables.

In the case of the Class 2 probes, clearly if the query graph contains a certain number of vertices labeled in a given way, the target graph may possibly contain a perfectly matching subgraph so long as it contains at least that many vertices labeled in the same way. Similarly, the way in which the Class 1c probes are correlated needs to be modified as well, since the vertices present in the query graph may have fewer incoming and outgoing edges than their corresponding vertices in a matching subgraph of a larger graph.

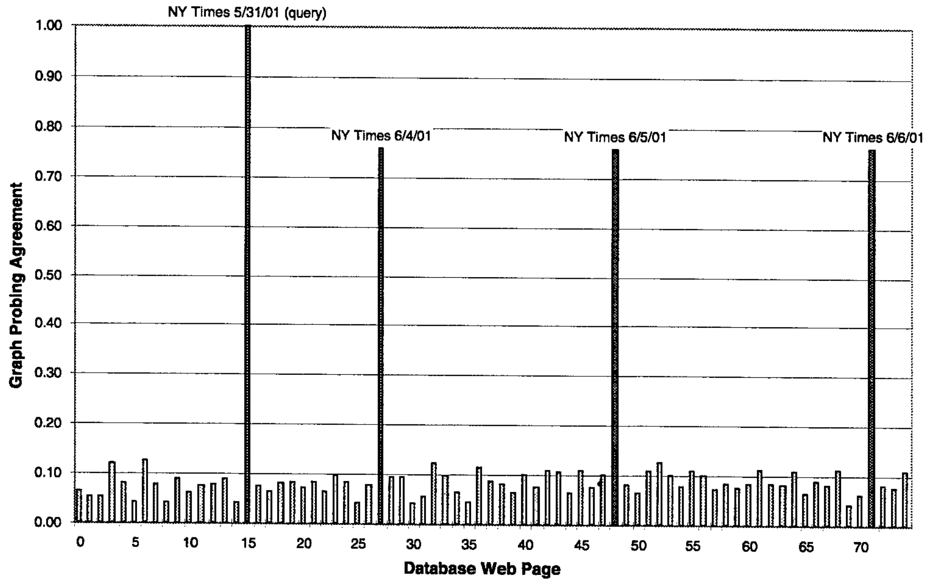


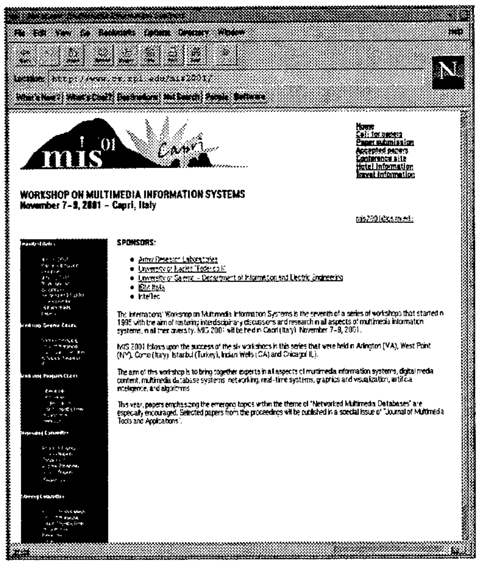
Figure 2: Graph probing results for Experiment 1 (full graph comparison measure).

5 Preliminary Experimental Results

This paper describes research that is still very much in progress. As we noted, we have previously implemented the on-line version of graph probing (Figure 1) to help in the evaluation of table understanding algorithms [5, 6]. However, the utility of graph probing in that specialized domain is not an assurance that it is an appropriate paradigm for searching databases for matches in a retrieval application, especially since the classes of probes we have at our disposal are also different here.

To begin examining some of these issues, we performed two simple experiments to test graph probing in an information retrieval setting. In both cases we used Class 1c and Class 2 probe sets. The first test examined the ability of the two classes to detect changes as the structure of a specific commercial Web page evolved naturally over several days. The second studied the subgraph matching problem by searching for a Web page that had been edited by deleting a significant portion of its content.

The small database consisted of 75 current WWW homepages collected from a variety of sources: commercial, educational institutions, personal, news organizations, and portal pages. Our parse graph generator is capable of recovering from the kinds of simple errors that often arise in real-world HTML (*e.g.*, missing end tags). The size of the resulting graphs ranged from a minimum of 60 vertices to a maximum of 1,204, with an average of 419 vertices. The probe set, generated automatically, consisted of a total of 219 Class 1c and Class 2 probes.


File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.cs.rpi.edu/mis2001/
What's New? What's Cool? Destinations Net Search People Software
mis '01 Capri
Home
Call for papers
Paper submission
Accepted papers
Conference site
Hotel information
Travel information

WORKSHOP ON MULTIMEDIA INFORMATION SYSTEMS
November 7–9, 2001 – Capri, Italy

SPONSORS:

- Army Research Laboratories
- University of Naples "Federico II"
- University of Salerno – Department of Information and Electric Engineering
- IBM Italia
- IntelTec

The International Workshop on Multimedia Information Systems is the seventh of a series of workshops that started in 1995 with the aim of fostering interdisciplinary discussions and research in all aspects of multimedia information systems, in all their diversity. MIS 2001 will be held in Capri (Italy) November 7–9, 2001.

MIS 2001 follows upon the success of the six workshops in this series that were held in Arlington (VA), West Point (NY), Como (Italy), Istanbul (Turkey), Indian Wells (CA) and Chicago (IL).

The aim of this workshop is to bring together experts in all aspects of multimedia information systems, digital media content, multimedia database systems, networking, real-time systems, graphics and visualization, artificial intelligence, and algorithms.

This year, papers emphasizing the emerging topics within the theme of "Networked Multimedia Databases" are especially encouraged. Selected papers from the proceedings will be published in a special issue of "Journal of Multimedia Tools and Applications".

Figure 3: Screen snapshot of the MIS'2001 homepage (http://www.cs.rpi.edu/mis2001/).

For the first experiment, we used as our query the May 31, 2001 homepage from *The New York Times* (http://www.nytimes.com/). This is a relatively complex page, its parse graph containing 1,089 vertices (for comparison, the graph depicted in Figure 4 is less than one-fifth this size). The results for using graph probing to compare this page to the 75 pages in the database are shown in Figure 2. The May 31 page is, of course a perfect match for itself, but also a very good match for the June 4, 5, and 6 homepages as well (the only other examples from *The New York Times* in the database). Clearly some sort of structural change in the page was made between May 31 and June 4. Conversely, a number of other regularly-updated Web pages we have examined show no structural changes from day-to-day, although obviously the content is constantly varying.

For our second experiment, we used the homepage for the MIS'2001 workshop, a screen snapshot of which is given in Figure 3. The corresponding parse graph, containing 193 vertices, is shown in Figure 4. To create the query, the page was edited by deleting the dark blue sidebar on the left side of the page. The parse graph for the edited page had 124 vertices and appears in Figure 5.

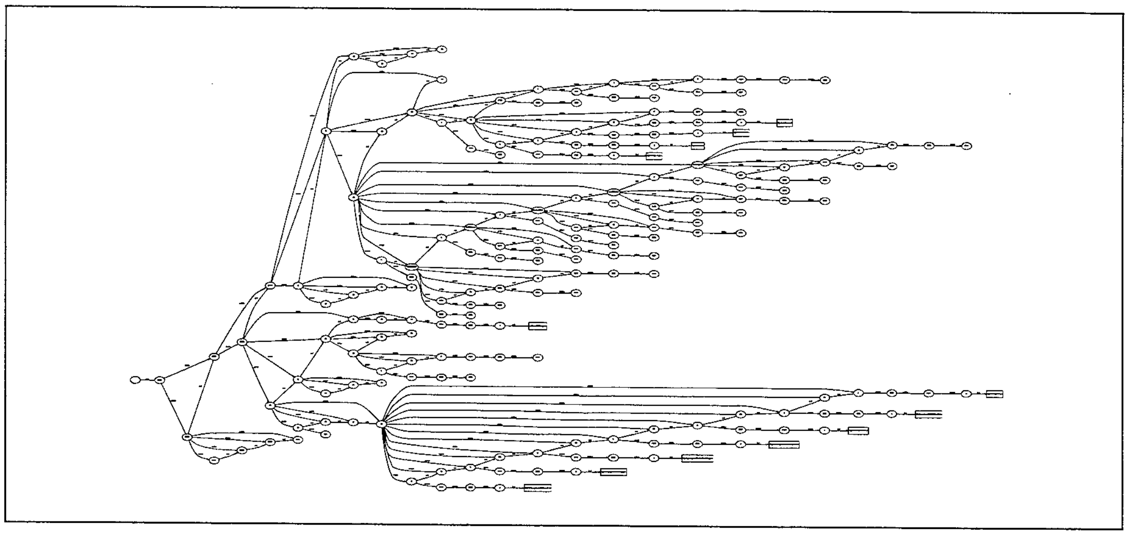


Figure 4: Parse graph for the MIS'2001 homepage (193 vertices).

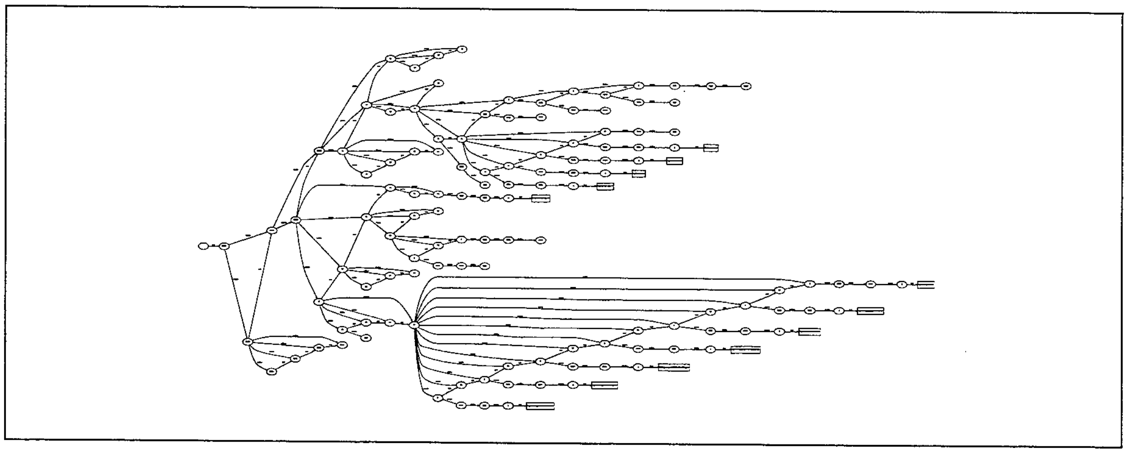


Figure 5: Parse graph for edited version of the MIS'2001 homepage (124 vertices).

The results for this experiment, using graph probing with the subgraph comparison measure, are shown in Figure 6. Here we can see that the two probe classes are able to distinguish the original page and the smaller, edited version from the rest of the database, but just barely. The current probes, which consider only structure (and high-level structure at that), need to be supplemented with new classes that are able to make use of content and other aspects of the page layout before the probing paradigm can be effective for the subgraph matching problem. This is work in progress.

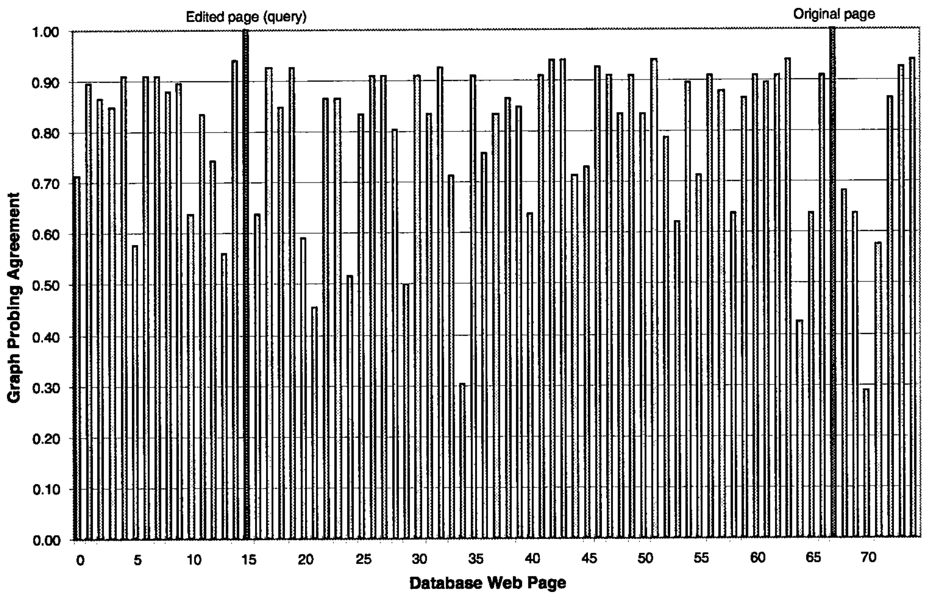


Figure 6: Graph probing results for Experiment 2 (subgraph comparison measure).

6 Discussion

In this paper we have described our initial efforts to adapt the graph probing paradigm to searching databases of graph-structured documents. We considered both the comparison of two graphs in their entirety, as well as the subgraph matching problem, and gave some preliminary experimental results.

Currently, our graph probing provides a measure of how similar two graphs are or how similar one graph is to some subgraph of another. It does not, however, produce a mapping from one graph to the other. Clearly, to extract information from semi-structured sources we need to recognize more than the fact that some matching is likely to exist; we must be able to identify the actual correspondence between the graphs (or between one graph and a subgraph of the other). At the very least, however, graph probing can be used as a filter to exclude graphs that could not possibly be a good match so that computationally expensive methods may be run on much smaller collections of potential candidates.

Another topic for future research is extending the formalism and bounds of Section 3 to the subgraph matching case.

References

[1] L. Babai, P. Erdös, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, August 1980.

[2] H. Bunke and B. T. Messmer. Recent advances in graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(1):169–203, November

1997.

[3] D. Dubois, H. Prade, and F. Sèdes. Some uses of fuzzy logic in multimedia databases querying. In *Proceedings of the Workshop on Logical and Uncertainty Models for Information Systems*, pages 46–54, London, England, July 1999.

[4] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(3):59–74, 1998.

[5] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. A system for understanding and reformulating tables. In *Proceedings of the Fourth IAPR International Workshop on Document Analysis Systems*, pages 361–372, Rio de Janeiro, Brazil, December 2000.

[6] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table structure recognition and its evaluation. In *Proceedings of Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging)*, volume 4307, pages 44–55, San Jose, CA, January 2001.

[7] N. Kushmerick. Regression testing for wrapper maintenance. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 74–79, Orlando, FL, 1999.

[8] M. Lazarescu, H. Bunke, and S. Venkatesh. Graph matching: Fast candidate elimination using machine learning techniques. In *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 236–245. Springer-Verlag, Berlin, Germany, 2000.

[9] D. Lopresti and G. Wilfong. Evaluating document analysis results via graph probing. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 116–120, Seattle, WA, September 2001.

[10] B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[11] M. Ortega-Binderberger, S. Mehrotra, K. Chakrabarti, and K. Porkaew. WebMARS: A multimedia search engine for full document retrieval and cross media browsing. In *Proceedings of the Sixth International Workshop on Advances in Multimedia Information Systems*, pages 72–81, Chicago, IL, October 2000.

[12] A. N. Papadopoulos and Y. Manolopoulos. Structure-based similarity search with graph histograms. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pages 174–178. IEEE Computer Society Press, 1998.

[13] T. Schlieder and F. Naumann. Approximate tree embedding for querying XML data. In *Proceedings of the ACM SIGIR Workshop On XML and Information Retrieval*, Athens, Greece, July 2000.

[14] G. Valiente and C. Martínez. An algorithm for graph pattern-matching. In *Proceedings of the Fourth South American Workshop on String Processing*, pages 180–197. Carleton University Press, 1997.

Indexing XML Documents: Improving the BUS Method[*]

Vincent Oria, Amit Shah, Samuel Sowell
College of Computing Sciences
New Jersey Institute of Technology
University Heights
Newark, NJ 07102-1982
oria@cis.njit.edu, acoolshah@usa.net, sowellms@mindspring.com

Abstract

In this paper we propose an indexing method for XML documents that improves the BUS method. The main difficulty in indexing XML documents is that the index must accommodate two different worlds that XML brings together: database and information retrieval. In the database world, indexing is based on the data structure, while indexing in information retrieval is based on the text content. One of the original XML document indexing methods is the Bottom Up Scheme (BUS). This method was then modified to create a better mechanism for updating structured documents. Despite the significant improvements made to the Bottom Up Scheme, the experiment still indicated a 240% storage overhead, indexing time of 4 hours and 43 minutes for 250 MB of documents, and query times as long as 6.46 seconds. The solution to minimizing the overhead introduced by the indexing is to reduce the number of terms to be indexed. The method proposed in this paper selects important words from the documents to be indexed, based on a user-defined dictionary. Furthermore, each node in the tree structure of the BUS method has a fixed number of children that causes complete tree reorganization from time to time. We propose a solution that avoids the reorganization of tree.

1.0 INTRODUCTION

The challenge/problem of searching documents has been around as long as electronic documents have existed. As Internet browsing has become more popular, information retrieval has increased in importance. Current Internet browsers search some portion of the documents stored on their servers, and return exact or similarity matches to the character string selected by the user. Advanced techniques can be added to allow the user to connect multiple strings, using logic such as "AND" and "OR". Given the lack of structure in HTML language, there have not been any widely used alternatives. The difficulty with the current approach is that the context semantics are not taken into account.

One alternative method that has been suggested for HTML and other unstructured electronic documents is the use of a standard ontology for a given field, such as medicine, computer software or oil drilling. Once the ontology is developed, software can be developed to parse documents that have been created using the standard ontology. To date, this approach has not received universal acceptance because of the time and effort it would take to produce an ontology, and the burden placed on the user to conform to the ontology.

The lack of semantic definition has been addressed by the creation of two languages with more structure: SGML and, more recently, XML. Both of these languages have a semantic structure that can be used to create and search documents. As more XML documents are created, using semantics for document retrieval is becoming a priority for computer users, especially in the

[*] This research is supported, in part, by the New Jersey Institute of Technology under the Grant 421130.

technical community. To date, two approaches, which incorporate semantics, have been developed to address information retrieval.

The first approach is to create document-specific software that contains algorithms able to search for very specific information in documents of a known structure and content. This method has been used, for example, by lawyers in Australia who have created an algorithm to look at a narrow set of court decisions to determine pertinent information [1]. In addition, they have recommended a standard ontology to improve search results. Scientific researchers have also created a product to index and retrieve scientific literature [2]. Using this method for scientific or judicial documents is appropriate for a limited setting, but it provides a narrow-focus search that produces less than desirable results when used for documents outside the original domain.

The second approach is to use structured documents and databases that allow semantic information retrieval, as the documents contain tags that can be semantically interpreted. SGML and XML languages both provide semantic tags. As Lee *et al.* stated in [3], "In order to perform structure queries efficiently in the structured document management system, an index structure that supports fast element access must be provided because users want to access any kind of document element in the database." They proposed two index structures that could be used to facilitate access to an element in the database - inverted index structures and signature file structures. These two structures have been the basis for recent indexing research.

As described in [4] and [5], signature files only index representative words or "record signature". Object signatures are stored in a bit format and compared to the query to determine matches. Using signature files greatly reduces the storage requirements and minimizes access time. The obvious drawback of using signature files is that very little information is included in the indexing system. Therefore, the user can only search on a limited set of criteria.

Inverted file structures solve the problem of limited search criteria created by the signature file. The inverted file structure, however, creates three new problems: significant increase in the amount of storage required for a document, a significant increase in the amount of time required to parse the document and index the elements at every level of the file structure, and an increase in the query time. To reduce storage requirements and improve query processing, Shin et al [6] developed the Bottom Up Scheme (BUS). This scheme reduces the storage requirements by using a general identifier (GID) for each element so that indexing occurs only at the leaf node (text) level. The BUS method was further modified to create a better mechanism for updating structured documents [7]. Despite the significant improvements made by the Bottom Up Scheme, the experiment still indicated a 240% storage overhead, indexing time of 4 hours and 43 minutes for 250 MB of documents, and query times as long as 6.46 seconds. Because BUS uses a complete tree structure including virtual nodes, significant updates require the restructuring of a large portion of the inverted file structure.

The intent of this paper is to show how the BUS indexing method can be used as a basis for an indexing method that will reduce the time involved to store, query and update SGML and XML documents, while reducing the storage overhead. The efficiency of updating the content and tree structure can be improved by modifying the BUS method so that a complete tree structure is not necessary. Storage time and space, as well as query time, can be improved by including a hierarchical data dictionary in the database. The hierarchical dictionary is a more flexible notion ontology that can range from a list of keywords to a complex ontology. The hierarchical dictionary includes all relevant concepts and terms for a given application domain. Although building an ontology can be a difficult task, the flexibility in our approach makes it practical: a list of domain keywords as seen at the end of every textbook is enough for a start.

2.0 RELATED WORKS

In the past years, several methods have been created to retrieve information from structured documents. Path encoding [8] is a technique that encodes each path from the root to the destination node in the hierarchy into a compressed form [7]. The index overhead is low, but the paper did not address the query evaluation.

Lee *et al.* [3] created an index that reduced the storage overhead, by indexing at all levels of a document. This method uses a complete tree where the parent element has a predefined maximum number of children (k). Any tree created would consist of real nodes indexed by keyword and virtual nodes to provide room for future elements. Each element is assigned a Unique element IDentifier (UID) using a left-to-right, top-to-bottom traversal that can be used to calculate the parent UID.Development of this method allowed an index to be created only at the parent level if an index term appeared in all of the leaf nodes of the tree to be indexed. The method enabled some queries of structured documents, but it did not provide a mechanism to determine index weights.

The Bottom Up Scheme (BUS) [6] built on Lee's work by creating a General element IDentifier (GID), which included the UID of the element and the level of the element in the tree. This makes it possible to include weight information. Although this improvement was significant, the major contribution of the BUS scheme is the reduction of indexing overhead by indexing only at the leaf nodes. During query evaluation, higher-level information can be determined by collecting information from the lowest levels of the document.

In order to more efficiently update the database indices following document updates, Jang *et al.*[7] extended the BUS method by using Oracle Index-Organized tables. Three postings were created: *content*, *structural*, and *attribute*. The Index-Organized tables improved the efficiency of index updates, and the postings allowed the user to query the document using content, structure, and attribute criteria.

2.1 Bottom Up Scheme (BUS)

The Bottom Up Scheme (BUS) [6] is an indexing method that minimizes storage overhead by indexing only the leaf nodes, and also provides relatively efficient techniques for querying structured documents. It is an extension of the indexing structure proposed by Lee [3]. As described above, Shin *et al.* were able to reduce overhead by representing a document as a complete tree with *k* children per parent. To identify each of the members of the tree, they also created the UID; the UID of parent node is determined using the following formula:

$$\text{Parent(UID)} = \frac{\text{UID} - 2}{k} + 1$$

Remember that in the tree, the UID is assigned following a left-to-right and top-to-bottom traversal. Since the number of children for a node is fixed, room is reserved for non-existing children called *virtual nodes* as opposed to *concrete nodes*. The BUS also introduced the concept of the General element Identifier (GID) by extending the UID concept to include (1) Document Number, (2) the UID of the element, (3) the level of the element in the tree and (4) the element type number. The document number distinguishes the element from elements contained in other documents, and the UID distinguishes it from elements within the document. The level of the element facilitates queries that allow the user to select the level at which terms are to be collected. The element type number facilitates queries when the user wants to retrieve information based on the structure of the document. The central concept of the BUS method is to maintain indexing information (the GID) only at the leaf nodes of a B+ tree structure. Information at higher-level nodes can be determined from the information stored in the leaf nodes at run time. If a user wants to get information at an intermediate level, term frequencies at the leaf nodes are automatically accumulated to the corresponding ones at the intermediate level [7].

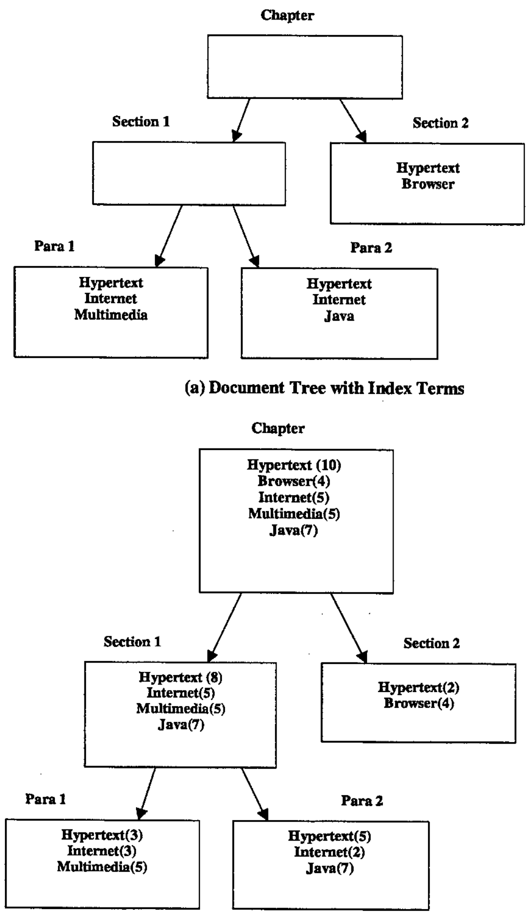


Figure 1: Bottom Up Scheme Query Processing

The BUS method executes indexing for each element at the text level by extracting all of the GID information and the frequency of a term appearing in the element. First, it scans the document, assigning a GID to each element; then it extracts terms and calculates their frequencies in each element at the text level [6]. The result is an inverted index with the B+ tree and the posting file. The Query Evaluation Procedure (QEP) of BUS first creates a set of accumulators corresponding to all of the elements in the document set. It then analyzes the query to determine which level and element type the user wants, and extracts the appropriate postings. Using these, QEP maps the posting UID to the parent UID and adds the frequencies into the accumulators until it reaches the level desired by the user. Figure 1a) shows a document structure with some terms at the leaf nodes and Figure 1b) presents the accumulators for a query at the root level.

2.2 Improved BUS Method

Jang et al. [7] improved the efficiency of updating a structured document by organizing the index into relational tables (Index-Organized Tables), especially designed for information retrieval and storage of indices, and implemented on top of the Oracle system. An Index-Organized table is an index that stores the terms and the auxiliary data in leaf nodes.

Indexing consists of preprocessing the original documents, extracting postings, and loading posting information into database tables. Preprocessing and posting extraction extract words from the documents, calculate the UID of the element, compute the term frequencies in each element, and associate a GID to the term. The data are saved in three postings: content, structural, and attribute. In addition to the GID information, the index word is saved in the content posting, the level and number of children are saved in the structural posting, and the attribute name, level and attribute value are saved in the attribute posting. Postings are loaded into Index-Organized Tables (Content Index Table, Structural Index Table, and Attribute Index Table), using SQL*Loader to increase the speed of the process.

Updates can cause changes to the element content and/or structure. When the content is changed, new indices in the Content Index Table replace old ones. When the elements are inserted and deleted, content typically changes also. Thus, insertion of a new element normally causes the replacement of indices in both the Content Index Table and the Structural Index Table for the new element, and possibly sibling elements. If the insertion of an element changes the value *k* (the maximum number of sibling elements in a tree), then the whole content and structure of the document should be updated. Deletion of an element causes the deletion of all postings related to the element, and may cause the update of content. As is the case with insertion, sibling elements may be affected.

The improved BUS method has the ability to retrieve information using the content, structural, or attribute index tables, individually. A content query uses a term and element name, and retrieves postings whose terms match the given term. A structural query retrieves elements that satisfy the condition imposed by the query. An attribute query uses the element name, and attribute name and value to return the document ID and the UID.

3.0 MODIFIED BUS METHOD (MBM)

As mentioned above, the improved BUS method implemented by Jang *et al.* still has significant drawbacks. The overhead is large, indexing and query times are long, similarity queries cannot be performed, and some updates require re-indexing the entire document. We propose the modified BUS method (MBM) of indexing and querying using a hierarchical dictionary in order to address the storage overhead, indexing time, querying time, and similarity query concerns; and a modification to the tree structure to eliminate the need for re-indexing the entire document during updates.

3.1 Hierarchical Dictionary and Concept Hierarchies

Given the reduction in storage overhead that BUS has achieved by indexing only leaf nodes, it is unlikely that there is another method that will greatly reduce the overhead while indexing the same number of terms. Therefore, the only choice is to reduce the number of terms to be indexed. The obvious concern with reducing the number of indexed elements is that the user will not be able to retrieve information on all words in the document. But are all the words in a document of the same importance? At the end of every textbook there is an index that lists all the important words on the topics covered in the book. Indexing the textbook based on the words in the index will certainly be sufficient to answer most of the questions related to the topics in the textbook. Another approach is to index only the instances of certain important element types. For a book, these might be the author, title, ISBN, publisher and copyright date. The solution presented in this paper uses the two approaches in conjunction.

Selecting in advance the terms to index will be more effective than indexing all the words in a document in addition to reducing the index overhead. This solution requires the application designer to provide some domain knowledge for each specific application. Although it requires some additional work for the designer, the flexibility in our notion of hierarchical dictionary makes the solution practical, as there is no need to have a complete ontology in order to start using such a system. The domain knowledge can range from a flat dictionary composed of a list of words, to a complex ontology. The dictionary starts with a list of words of idioms of interest to the application. The application developer can then build a concept hierarchy with a list of keywords provided as leaves of the hierarchy, in order to obtain a complete hierarchical dictionary. An example of a hierarchical medical dictionary, based on an ontology with a small section of topics that might be chosen, is shown in Figure 2. Using this second method of indexing, only words in the specific dictionary and relevant concept hierarchies would be indexed, greatly reducing the total number of words stored in the database and - therefore - indexing time. The use of the first and second methods together greatly reduces the amount of storage overhead caused by indexing of the entire document, incorporates the ontology concept, increases the number of search criteria over signature files, and provides flexibility for the user.

The hierarchical dictionary provides an added benefit when used during the query process. When the user writes a query, the dictionary can be used to find all indexed words related to the user topic. This allows the user to search for a family of words based on the dictionary hierarchy. For example, if the user wants to find all documents that are related to "dentistry", with an index based on Figure 2, the retrieval process will return all documents that contain "dentistry", "orthodontics" and "periodontics". The overall effect of using the hierarchical dictionary is that the user can retrieve information by specific indexed keywords, families of keywords, or similarity searches at any level in the document. The level determination is performed using the weight accumulation query features of the BUS system. Despite the increased power of MBM queries, the queries should still be performed more quickly than the BUS or improved BUS methods, due to the relatively small number of indexed terms.

3.2 Avoiding Re-Indexing on Updates

As discussed in the Improved BUS section, the BUS system maintains a complete tree with a maximum number of children, *k* [6]. Whenever an update is made that causes the number of children for a particular parent to exceed *k*, then the document must be completely re-indexed and each element must be given a new UID to allow the creation of a new complete tree. Adding the parent ID to each element GID will eliminate the need to maintain a complete tree with a fixed number of maximum children. Using the MBM method, as each element is indexed, the next sequential UID can be assigned. Therefore, updating with MBM never requires the entire document to be re-indexed.

Figure 3a, borrowed from [6], shows a sample of a BUS Document Tree - with the solid boxes representing the actual nodes and the dashed boxes representing the virtual nodes. The BUS method assigns UIDs starting at the top and traversing the tree left-to-right, including virtual nodes. If this document is updated to include a real S3, the UID has already been assigned, so there is nothing else to do than making the virtual node S3 visible. If, however, the document is updated to include an S4 element, then the entire document has to be re-indexed using a tree with 4 maximum children. MBM does not have a restriction on the maximum number because each node explicitly stores the parent UID (Figure 3b). Thus, when a new element is added, it is assigned the next available UID and re-indexing of the entire document is never performed. As was the case for the BUS method, we chose a relational database with hierarchical tables developed by Oracle.

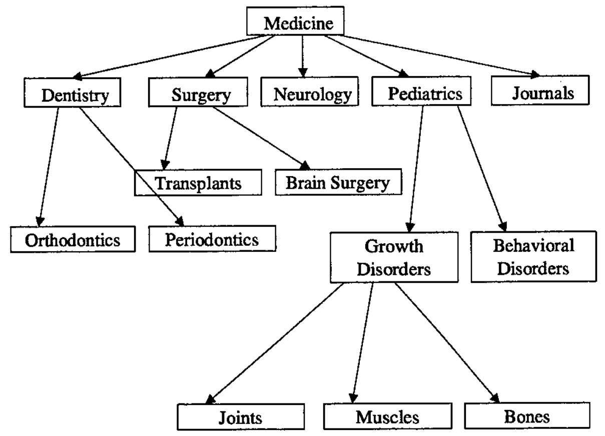


Figure 2: Sample Medicine Document Set Dictionary

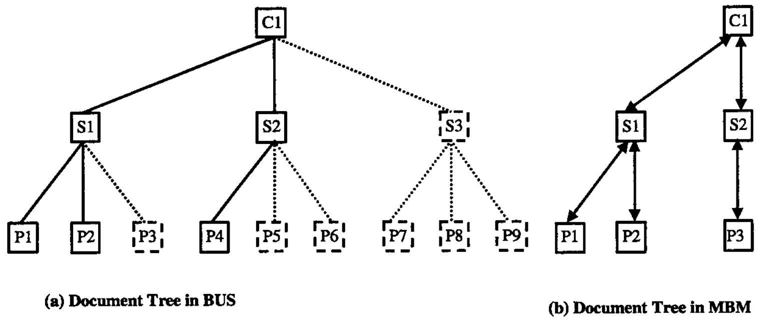


Figure 3: Document Trees in BUS and MBM

4.0 IMPLEMENTING MBM

The first step in indexing a document set is to load the user-defined dictionary or dictionaries. Each dictionary document is parsed, and the dictionary structure is extracted and stored in the Keywords and Concept_Relationships tables. The leaf concepts in the Concept_Relationship table are keywords from the Keywords table, on top of which the hierarchical structure of the dictionary is built.

Once the dictionary has been loaded, the document set DTD needs to be loaded, if available. If not, reviewing the available document set can create the DTD, and then it can be loaded. The information from the DTD is stored in the DTD and DTD_Attributes tables. In the DTD table, Element_Type_Code, Parent_Element_Code, Element_Name, Multiplicity, and Index_By_Dictionary are stored using Element_Type_Code as the primary key for the table. The Element_Type_Code is the same as that used by the two BUS methods - it defines the element type from the DTD. The Parent_Element_Code is the element code of the element if it exists. Multiplicity describes whether the element is required/allowed to have zero, one or more instances when used in the document. Index_By_Dictionary is a Boolean attribute that is 1 if the element is type is to be used when retrieving information by using the dictionary. In the DTD_Attributes table, Element_Attribute and Required are stored with Element_Attribute as the primary key and Element_Type_Code as a foreign key. Element_Attribute is one of the component parts that make up an XML element. For instance, "Last_Name" might be part of the element "Person". The attribute "Required" refers to whether the given attribute is required to be specified when the given element type is used in the document.

Information from the document set can be parsed, extracted, and identified once the dictionary, concept hierarchy and DTD have been loaded. Before indexing, the dictionary to be used for indexing needs to be identified. Indexing consists of preprocessing the original documents, extracting postings, and loading posting information into database tables in a similar manner to the improved BUS method [7]. Preprocessing and posting extraction extract words from the documents based on comparison to the DTD, hierarchical dictionary, and the concept hierarchy.

As the documents are parsed, information about each document is determined and posted at three levels: document level, element level and element attribute level. Document level information includes the doc_id, doc_data, doc_length. Doc_id is the unique number that identifies each document and corresponds to DID in the BUS system, doc_data is the title of the document, and doc_length is the length of the document in words. Element level information includes the element_id, Element_Type_Code, parent_element_id, level, and frequency. The element_id is the unique number that, along with doc_id, identifies each element. Element_Type_Code is determined by comparing the element type to the DTD and determining the number. The parent_element_id is the element_id of the parent, and is used to parse the tree during queries. Level describes where in the document the element is found. Frequency is the number of times the element appears at the given level. Element attribute information includes the attribute and value. Attribute is the type of attribute, and value is the actual number/characters in the document. Using a previous example, an attribute of person might be last_name and have a value of Johnson.

The last task of indexing is to populate the tables of the database. The first table to be filled is the elements table, which contains element_id, doc_id, Element_Type_Code, parent_element_id, and level, with element_id as the primary key and doc_id as the foreign key. The next table to be filled is the attributes table, which contains attribute, element_id, doc_id, and value with attribute as the primary key; and element_id and doc_id as the foreign keys. The last table to be filled is the element_keywords table, which contains keyword, element_id, doc_id and frequency, with keyword, element_id, and doc_id as the foreign keys. The E-R diagram resulting from indexing and the loading of the DTD data dictionary is shown in Figure 4.

5.0 MBM QUERY PROCESSING

When a user defines a query, several steps need to be executed to return the result. First, the query must be parsed to identify parts of the database participating in the query. The Concept_relationships, Elements, Element_Attributes, and Element_Keywords tables may all be needed to execute the query properly. If keywords higher in the dictionary hierarchy are needed, then the query has to be expanded to include all of the dictionary keywords at or below the level of the user query. MBM uses a method similar to the BUS method for processing the SQL query [6]. First, the query creates a set of accumulators corresponding to all indexed elements in the document set. Secondly, it determines which level and element type is wanted and compares desired level to actual level. Thirdly, it maps the element_id to the parent element_id, and the frequency is added into the accumulator. At the end, the process has summed all of the frequencies of the descendant element to the accumulator corresponding to the user level element. The result that is returned is a set of documents that satisfy query criteria.

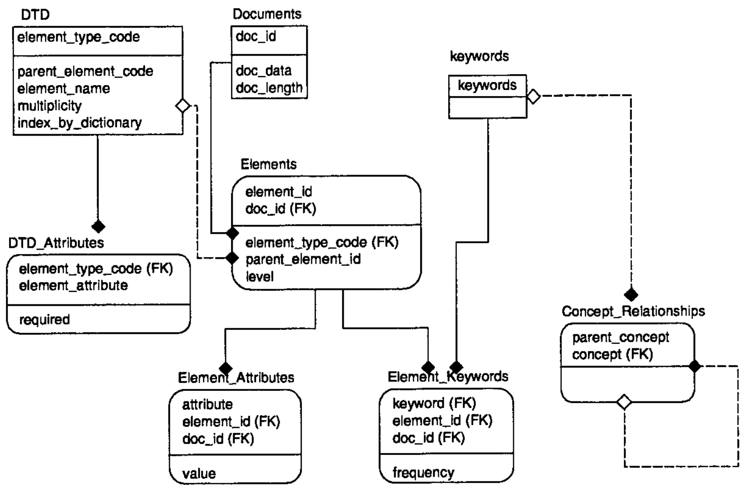
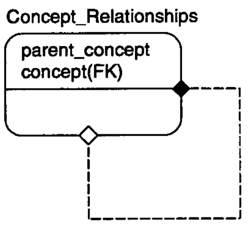


Figure 4: MBM Index-Organized Table E-R Diagram

5.1 Hierarchical Queries

The classical approach to resolving hierarchies is to employ the recursive approach. That is, to find all children of a particular node, we visit each child and, recursively, visit their children. The standard SQL for relational databases is not amenable to hierarchical queries. Yet, there are numerous examples of hierarchies modeled as relations. To resolve such hierarchies, Oracle provides some additional clauses such as CONNECT BY, PRIOR, and START WITH which can be used in conjunction with standard SQL.



For example if we wanted to recover the entire subject hierarchy under Pediatrics (see figure 2), we could have a query as follows

```
SELECT subject, parent_subject, level
FROM Concept_Relationships
CONNECT BY PRIOR parent_concept = concept
START WITH subject = 'Pediatrics'
```

Figure 5: Using the Hierarchical Dictionary in a Query

The above-mentioned query would give the result set shown in Table 1. Note that level is not a column in Table1 but an Oracle reserved word, which is treated as a function. Similar techniques are used for resolving the various hierarchies, such as Element Type Hierarchies.

Concept	**Parent**	**Level**
Pediatrics		1
Growth Disorders	Pediatrics	2
Joints	Growth Disorders	3
Muscles	Growth Disorders	3
Bones	Growth Disorders	3
Behavioral Disorders	Pediatrics	2

Table 1: The Hierarchy From the Subject "Pediatrics"

5.2 MBM and Updates

Content and structural updates are easy to accomplish using MBM. When a document is modified to create new elements, the section of the document that has been modified needs to be re-indexed, and new elements are assigned new element_id numbers with the associated information - including parent_element_id - discussed earlier in this paper. On deletion, the process is simpler. Re-parse the affected section, deleting table entries that no longer exist. This method is much simpler, and therefore much quicker, than the document update method used by the improved BUS system.

6.0 CONCLUSION AND FUTURE WORKS

The BUS and improved BUS methods made a significant contribution to structured document retrieval using semantics. These methods, however, still are not satisfactory as they induce long indexing time, long query time, large storage overhead, inefficient update method, and the inability to perform similarity queries.

This paper improves the BUS method by proposing to index fewer DTD elements and terms. Only the terms that appear in a user-defined hierarchical dictionary, and selected DTD elements, are indexed. This leads to a lower storage overhead, and shorter indexing and querying times. The hierarchical dictionary can range from a list of keywords, such as keyword indexes seen at the end of textbooks, to an ontology. When the dictionary is hierarchical, it is possible to pose some similarity queries. The query on an internal node of the hierarchical dictionary is enlarged to include all the terms in the sub-tree that start with the query term. Unlike the BUS, a node in the index tree we use does not have a fixed number of children in order to avoid a complete reorganization of the tree in case of heavy updating. Although our notion of a hierarchical dictionary is flexible enough to range from a flat list of keywords to a complex ontology, providing the dictionary is still a tedious task. But in some applications like education that we are currently applying our method to, it is easy to find lists of keywords at the end of textbooks.

There is work that remains to be done, based on MBM. We are currently conducting experiments, taking into account different types of document styles, to determine what improvement may be possible. The results will be included in subsequent publications.

REFERENCES

[1] James Osborn and Leon Sterling. JUSTICE: A Judicial Search Tool Using Intelligent Concept Extraction. In *ICAIL* 99, Oslo, 1999.

[2] Steve Lawrence, Kurt Bollacker, C. Lee Giles. Indexing and Retrieval of Scientific Literature. In *CIKM* 99, Kansas City, November 1999.

[3]Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon and P. Bruce Berra. Index Structures for Structured Documents. In *DL* 96, Bethesda, 1996.

[4] Yangjun Chen and Karl Aberer. Layered Index Structures in Document Database Systems. In *CIKM* 98, Bethesda 1998.

[5] Seyit Kocberber and Fazli Can. Compressed Multi-Framed Signature Files: An Index Structure for Fast Information Retrieval. In *SAC* 99, San Antonio, 1999.

[6] Dongwook Shin, Hyuncheol Jang and Honglan Jin. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. In *DL* 98, Pittsburgh, 1998.

[7] Hyunchul Jang, Youngil Kim and Dongwook Shin. An Effective Mechanism for Index Update in Structured Documents. In *CIKM* 99, Kansas City, 1999.

[8] J. A. Thom, J. Zobel and B. Grima. Design of Indexes for Structured Documents. In *CITRI*, 1995.

Multimedia reporting: building multimedia presentations with query answers

Augusto Celentano and Ombretta Gaggi

Dipartimento di Informatica, Università Ca' Foscari Venezia
{auce,ogaggi}@dsi.unive.it

Abstract. A *multimedia report* is a multimedia presentation which integrates data returned by one or more queries to a multimedia database, thus extending the concept of report familiar in traditional structured databases. In such a scenario information retrieval consists in building a continuous presentation in which the retrieved data are located, connected, synchronized and coherently presented to a user.
We discuss modelling of multimedia reports in terms of data co-ordination and synchronization, based on a synchronization model we have defined for specifying complex multimedia presentations. As in a report the user can browse the returned data without loosing consistency, in a multimedia report moving along the presentation time requires appropriate synchronization to be guaranteed.

1 Introduction

Multimedia databases are similar to structured databases concerning basic operations: classifying, storing and retrieving data through algorithmic procedures and interactive interfaces. Nevertheless, the presence of different media types adds new facets to the operations and increases the complexity of data management.

In this paper we approach a quite traditional problem in the new perspective of multimedia data retrieval. The problem is basically the construction of a *report* on a set of data, and can be stated as follows: given a set of multimedia data, a retrieve task consists in building a continuous presentation in which the retrieved data are located, connected, synchronized and coherently presented to a user.

Reporting is one of three basic access modes to a data repository, the other two being browsing and querying. *Browsing* means to access a data repository along a priori undefined paths according to an estimate of relevance that the user formulates as he or she proceeds in the exploration of the repository. *Querying* means to identify relevant information according to precise and pre-defined selection criteria based on the content of the information.

Accessing data by reporting means that the retrieved data are meaningful as a collection, and the relationships among data items are perceived as relationships among aggregations which have a meaning in the application domain. Also the presentation schema of the report suggests the user a way of reading it and adds further semantics to the data.

When the operating environment evolves from text-only databases to complex and distributed multimedia repositories, reporting must be extended to face issues like media delivery and synchronization, channel management, user interaction, and cannot be effectively performed without a suitable model for describing the multimedia presentation which constitutes the report itself.

The problems of automating the construction of multimedia presentation have been approached by several authors, that we review in Section 2. We focus our discussion to data co-ordination and synchronization, based on a media synchronization model we have defined for specifying complex presentations made of continuous media.

We assume the World Wide Web as the surrounding environment of our discussion. As a first consequence, we are concerned with scenarios in which media items might be delivered independently, possibly by several servers, and must be synchronized at the user client site. Then, since data instances are not known in advance, we are concerned with a data integration and synchronization model which is based on data classes and types rather then on data instances, a quite

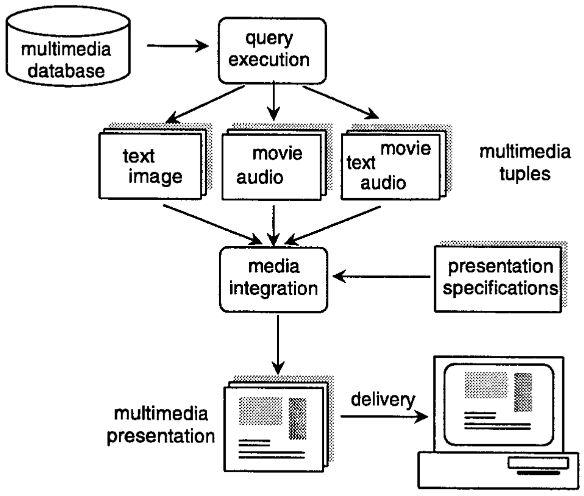


Fig. 1. Overview of the multimedia reporting process.

different situation with respect to traditional multimedia authoring and integration environments. Last, we are interested in building complex and interactive multimedia presentations. Therefore, we must define how different types of data can be presented together and how user actions must be handled in order to maintain the whole presentation coherent.

Figure 1 summarizes the process of building a multimedia presentation by querying a multimedia repository. Query execution returns a set of multimedia tuples whose components belong to different media types. In each tuple instances are homogeneous[1] therefore the specifications for media playback are compatible over all the data instances of a specific tuple type. The presentation specifications describe, among other information, how media items are integrated; the specifications describe how media items behave when events related to their playback happen. Master-slave relationships are established among media types, so that when the overall presentation is delivered to the user, its behavior is dictated by the dynamics and the timing of a master medium, usually a video clip or an audio track. Once the presentation is started, the master medium generates events when it starts, ends, or when the user interacts, e.g., by stopping playback. Such events propagate to other media, which in turn generate other events, determining the dynamic evolution of the whole presentation.

In this paper we are concerned only with the temporal and synchronization aspects of integration of the returned media items. We therefore do not face query specification and execution, nor we deal with layout specifications of the generated presentation. In Section 6 we shall discuss some open issues about multimedia querying.

The paper is organized as follows: in Section 2 we review the relevant literature by comparing the different goals and approaches taken. Section 3 introduces the functionality of multimedia reporting in terms of co-ordination and synchronization among the media objects returned. In Section 4 a suitable model for multimedia synchronization is presented, while Section 5 illustrates how such a model can be applied for specifying multimedia reports. Section 6 discusses consistency issues related to multimedia integration, and Section 7 presents concluding remarks.

2 Related work

Automation of the production of multimedia presentations has been approached in recent years from several points of view. All the approaches have the common goal of building in a more or less automated way a multimedia document that integrates different media objects extracted from a database or from document segments.

[1] In Section 6 we shall argue about this assumption.

SQL+D [1, 2] is an extension to SQL which defines the presentation properties of a multimedia query result. Given a multimedia database, an SQL+D query allows users to specify in the query a screen layout and time intervals to show the answer. In addition to SELECT-FROM clauses, DISPLAY-WITH clauses define screen areas (called *panels*), in which groups of retrieved media items are placed with specified relative positions. A SHOW clause defines the temporal behavior in terms of timed sequences of returned instances display.

The Cuypers system [15–17] is a prototype multimedia transformation environment supporting semi-automated assembling of multimedia documents according to a rich structural and semantic annotation based on XML. The annotation allows different processing steps concerning semantic structure, constraints satisfaction and final form presentation, which occur in multimedia authoring, to be integrated in a single execution stream.

In [10] the authors present a methodology for automated construction of multimedia presentations. Differently from the works reviewed above, the paper does not deal with query or retrieval environments, but focuses on the semantic coherency of a multimedia presentation in terms of inclusion and exclusion constraints. Given a set of multimedia document segments, the authors discuss how a user selection of some segments should be completed or modified by including additional segments based on semantic consistency of their content.

In [3] a system for the automatic generation, integration ad visualization of media streams is described. The authors view teaching and learning experience as a form of multimedia authoring, so the solution proposed are strongly oriented towards the educational domain, although this system can be applied to different fields. The paper describes a method to integrate different media streams, such as a video captured during a university lecture and its audio track, through the use of different levels of granularity, and of one particular solution to each level. A timeline "road map" of the lecture, marked with significant events, is proposed for the visualization of multiple streams.

Delaunay^{MM}[8] is a framework for querying and presenting multimedia data stored in distributed data repositories. Media objects returned by a query are inserted into a multimedia presentation. Delaunay^{MM} uses profiles to design user-defined layout of a document and ad hoc querying capabilities to search each type of media item. The authors do not address any formal model for the specification of temporal synchronization of different objects.

In most of the reviewed systems the authors assume (coherently with the examples provided) that the data items retrieved or selected for building the presentation are in some way predictable and homogeneous, i.e., they can be combined in temporal sequences or spatial layout with an *a priori* specification. This could be stated as an explicit requirement in order to be able to build presentations which are coherent for a user, and is almost true for static items like text and images. However, dynamic media like video and audio may need additional specifications of temporal behavior in terms of co-ordination and synchronization, that we shall discuss in this paper. The assembly of a multimedia dynamic presentation thus requires suitable models for specifying structure and temporal scenarios of hypermedia documents.

Amsterdam Hypermedia Model [11–13] was the first serious attempt to merge document structure with temporal relations. AHM distinguishes between atomic and composite components: the former are simple media elements, like a video file or a text page, and the latter are composition of different objects grouped together according to synchronization relationships. Media items are associated to channels which represent the devices needed for their playback.

SMIL[19], Synchronized Multimedia Integration Language, is a simple markup language defined as a W3C recommendation. It is an XML application to describe temporal behavior of a multimedia presentation using tags. Synchronization is achieved through tag `seq` to render two or more objects one after the other, and tag `par` to reproduce them in parallel. Using attributes it is also possible to play segments inside the time span of an object.

In [7] a formal framework for verifying temporal synchronization of a presentation is defined, with the goal of determining whether a multimedia presentation is synchronized or amenable to synchronization.

Madeus[14], an authoring and presentation tool for interactive multimedia documents, describes synchronization among different objects through the use of timing constrains based on particular events, like the beginning or the termination of a media file. Temporal information is represent through a direct acyclic graph, and layout is specified by spatial relations like *align* or *center*.

In [18] the authors discuss a system for authoring and formatting hypermedia documents, named HyperProp. It uses composition to represent spatial and temporal information. HyperProp provides the structural views to graphically browse and edit the logical structure of a document, the temporal view to represent objects along a timeline and the spatial view to formatting objects layout.

In previous works [4, 5, 9] we have discussed the problem of authoring and navigating hypermedia documents composed of continuous and non continuous media objects delivered separately in a Web-based environment. We have introduced a formal model which defines a static structure and synchronization relationships among media objects belonging to a same presentation. Temporal relationships between different media are described according to the event-based approach. In [6] we have applied the model to the identification of the scope of a query answer in a set of multimedia presentations. More details on the model are given in Section 4.

3 Multimedia reporting

Let us suppose a fashion Company wants to offer Web users the possibility of looking at a personalized catalog of the last collection in the shape of a multimedia presentation performing a virtual fashion show. A repository stores multimedia information about the fashion collections. The user can query the repository selecting the set of data which set up the virtual fashion show.

We do not enter into details about the visual richness of such a show, because it is out of the scope of this paper. We simply note that the result can be built on several types of data: for example, a set of pictures which portrait models wearing the selected dresses, or a set of movie clips which play parts of real fashion shows in which the selected dresses are shown. Text pages can describe the collection items from a stylistic point of view, and other text documents could provide details about tissues, techniques, commercial information, and so on. In order to complete and make more pleasant the presentation, a soundtrack could be played with songs or spoken comments.

Can we call such a presentation a "multimedia report"? Indeed, several aspects of text-based reports are present also here. The simplest multimedia presentation is a *slide show*: an ordered sequence of images displayed sequentially with appropriate timing and visual transitions, possibly with associated text documents. Such a presentation is similar to a report on an alphanumeric database because it exhibits a high degree of homogeneity in handling the components of the presentation: it is an organized collection of data, the overall structure reveals a recurring pattern, the transitions between data instances are perceivable like the breaks between groups in a report. Basically a slide show is a report translated from a linear space dimension to a linear time dimension.

Some of the systems reviewed in Section 2, e.g. SQL+D, can generate multimedia reports of such kind. Sequence, transitions, timing and spatial layout can be described by DISPLAY-WITH-SHOW clauses much as report sections are described by constructs of a reporting language. We can extend this simple scenario by introducing dynamic media which require complex synchronization specifications, e.g., by introducing movie clips and a soundtrack possibly made of different songs. In this case two data sets must be merged, and the transition between instances of the first set do not occur necessarily at the same time than in the second set, except for trivial cases. The slide show metaphor is no longer sufficient to describe such a scenario.

As an example, let us suppose that the presentation integrates a collection of dresses in the shape of movie clips coming from different fashion shows. Text pages describe the dress models, and for each fashion show a different song has to be played. A dress change (therefore a movie change) in the same fashion show, however, must not cause a song change.

This presentation is built on three different data types: the movie clips, the soundtracks, and the text documents associated to the movies. They can be returned by different queries as long as the instances of movies and texts are related by some foreign key correspondence[2].

[2] how such queries can be formulated is out of the scope of this paper

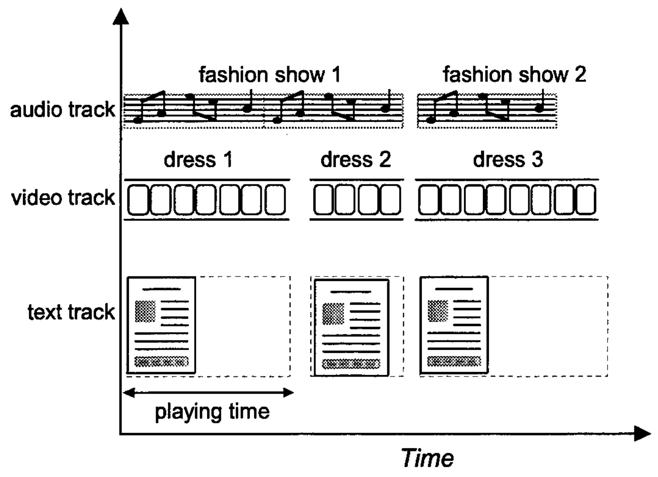


Fig. 2. Timeline of the example presentation.

We could attempt to describe such a presentation according to a timeline-based description like in Figure 2. Indeed, we cannot design such a timeline because we do not know how many objects will be returned by the query. Therefore we cannot define the transition times between instances. We do not even if text instances and movie instances are retrieved consistently, i.e., if for each movie there is a corresponding text page. Such a lack of information does not allow us to rely on models like the ones reviewed in Section 2. These models provide timing specification in terms of media sequencing and duration. In our example, moving from one fashion show to another causes not only a different movie to be played, but also a different soundtrack to be played. This event cannot be described in terms of media properties, but concerns the dynamic structure of the whole presentation

We need therefore to translate a representation based on a timeline into a *sequence of events* whose occurrence makes the presentation to evolve. In order to achieve this result we need a synchronization model that is able to define the relationships between the presentation of several media elements, defining also how the playback channels (e.g., the windows) are used and released.

Before describing this model we go someway further in order to justify the kind of synchronization relationships we shall introduce. The report composition model must allow us to go from a temporal description based on a timeline, like the one in Figure 2, to a description based on events like the one illustrated in Figure 3[3]. An event based description is more suitable for a Web environment since synchronization is not required to be fine-grained, but occurs only at specific events such as the beginning or the completion of a download, or the start and the end of a streaming medium, or a user action.

The model we have defined relies on relationships like "play medium B when medium A stops", which do not require to know in advance the duration and the physical features of the media involved, which are not known when the query is formulated and the presentation schema is defined.

4 Modeling synchronization in multimedia presentations

In previous works [4, 5, 9] we have proposed a model to describe synchronization among components of a multimedia presentation with reference to a distributed environment like the World Wide Web. The model defines relationships between media instances. It is however well adapt to describe relationships between classes, which in our scenario describe intensionally the query results.

Static components such as text and images are called generically *pages*; dynamic components are video and audio files. A hierarchical structure is defined which describes the overall presentation

[3] details of the figure will be described in next section

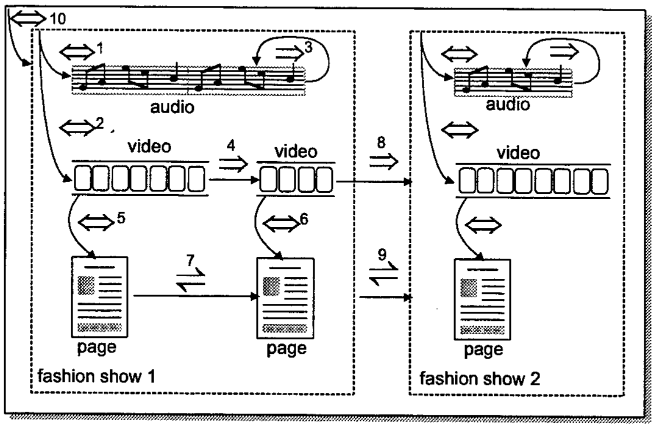


Fig. 3. Synchronization relationships between media items.

organization. The continuous media which constitute the main module content make up a *story*, which is composed of a sequence of continuous media files, audio and video, which are called *clips*. Clips are divided into *scenes*. A scene is associated to a (set of) static documents which are presented together. A clip with its scenes and associated pages build up a *section*.

Media objects require resources to be played. A *channel* is a virtual device allocated to a document media component, e.g. a window. Generally speaking, a channel is the set of resources a media object needs for its playback. Different media type require different channel types. A channel is *busy* if an active object is using it, otherwise it is *free* and can be used by another media object of the same type. Since static media do not evolve in time, they have an unlimited time extent, i.e., they holds a channel until forced to free it.

Synchronization is achieved with a set of primitives which define objects behavior during presentation playback and channels utilization. The events to which media react can be internal, like the beginning or termination of an object playback, or external, like a user action. We have defined five synchronization relationships.

- *A plays with B*, denoted by the symbol $\Leftrightarrow$ ($A \Leftrightarrow B$), to play two objects in parallel with object A acting as the master one; when it comes to end, object B play is also terminated.
- *A activates B*, denoted by the symbol $\Rightarrow$ ($A \Rightarrow B$), to play two objects in sequence; the end of object A causes object B to start playing.
- *A is terminated with B*, denoted by the symbol $\Downarrow$ ($A \Downarrow B$), to terminate two objects at the same time as a consequence of a user interaction or of the forced termination of object A.
- *A is replaced by B*, denoted by the symbol $\rightleftharpoons$ ($A \rightleftharpoons B$), to force the termination of object A so that object B can use the same channel.
- *A has priority over B with behavior* α, denoted by the symbol $\overset{\alpha}{>}$ ($A \overset{\alpha}{>} B$), to stop (if $\alpha = s$) or pause (if $\alpha = p$) object B when the user activates object A. This relationship describes the behavior of objects related by hyperlinks, that require to pause or stop part of a presentation in order to allow the user to focus the attention on the document at link destination.

Figure 3 shows an example of such relationships for the module described by the timeline of Figure 2. Relationships 1 and 2 state that the first audio track and the video clip showing the first dress start when the presentation of the first fashion show is started. By relation 3 the audio track is repeated as long as the presentation goes on. In this way we need not to be concerned with its length. Dresses are presented one after the other as described by relation 4 between the video clips. At the beginning of the first video clip, the first static page is presented, as described by relation 5, while the second page is displayed when the second video clips is played (relation 6). Since the two pages are displayed in the same window, the second one must replace the first one in the channel usage. This behavior is described by relation 7. The presentation of the first fashion show ends when the last video clip ends. Then, the second show is started, which takes the

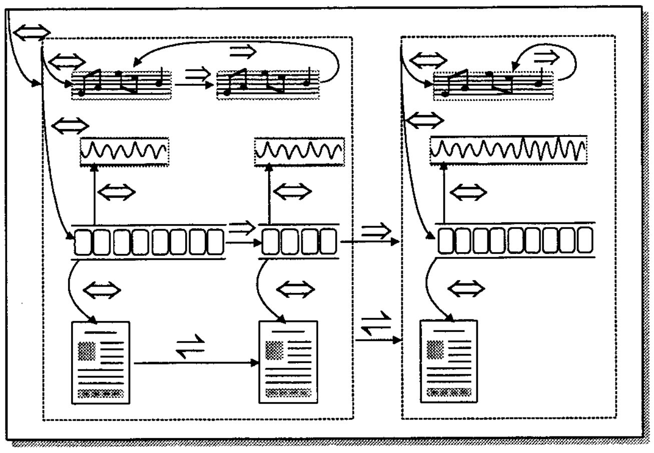


Fig. 4. A Synchronization schema for a more complex multimedia report.

resources that were allocated to the first part of the presentation. This is described by relations 8 and 9. The second fashion show proceeds in the same way, as well as the remaining ones. Relation 10 defines that the whole presentation starts with the first fashion show.

The whole presentation is structured in two levels, as revealed by the dotted regions they enclose media instances belonging to returned query results related to the same fashion show. The hierarchical structure allows the definition of environments in which some temporal relationships are derived by induction. For example, relations 1, 2 and 10. of Figure 3 are induced by the hierarchy between the whole presentation, its models (i.e., the different fashion shows), and the module components, therefore need not to be identified by the presentation designer, and introduce some degree of automation in multimedia authoring. The reader is referred to the cited previous works [4, 5, 9] for a more complete discussion of the model.

5 Presenting query results

The presentation of retrieved information can be described according to dynamic synchronization relationships, in order to present the user a coherent playback of the different media items involved. Let us modify and enrich our example, defining a multimedia presentation in which four kinds of media are involved, retrieved by queries from several related databases: movie clips, which portrait models with selected dresses; soundtracks, which play continuously in the background; text pages which describe some features about the portrayed dresses; and voice comments which introduce the different items of the fashion show, one short comment for each item. Also in this case the resulting presentation is organized along a hierarchical decomposition in two levels, the higher level defined by the aggregation of movie clips belonging to the same real fashion show (a group), the lower level defined by the sequence of movie clips portraying different dresses displayed in the same real fashion show. Figure 4 describes an instance of this presentation schema. In this case the voice comments are activated by the video clips, so that each clip starts a different comment. The soundtrack is still activated by the module which includes all the video clips of a specific fashion show, but executes several songs in sequence according to their own timing.

It is important to note that Figure 4 cannot describe *exactly* a presentation schema, since it has to be modelled intensionally and not extensionally, being unknown at presentation definition time the instances that will be returned by the query. It has the purpose of making evident what relationships need to be established among the returned media item in order to build a coherent view for the user.

The synchronization schema of this presentation is based on the following properties.

- The movie clips are the "master" section of the presentation. They give the overall timing to the presentation, because each of them defines the time a text page is displayed.

- The soundtrack is made of songs which are played one after the other within the same group. The time when a transition between two songs occurs is not related to other media, and depends only on the songs length.
- The voice comment is started by the movie clip to which it is associated.
- The whole presentation is a self-contained structure which begins execution by starting the execution of the first group, then letting each group start the following one upon completion. A group, in turn, starts the movie clip and the background soundtrack[4]. Starting from the presentation activation, every media item is played upon occurrence of events in the set retrieved by the queries, as described by the synchronization specification.

The constraints on media instances returned by the queries can be expressed by the following relationships. For each module:

$$\begin{array}{ll}
soundtrack_{k,j} \Rightarrow soundtrack_{k,j} & \forall j \\
video_{k,i} \Rightarrow video_{k,i+1} & \forall i \\
video_{k,i} \Leftrightarrow voice_{k,i} & \forall i \\
video_{k,i} \Leftrightarrow textpage_{k,i} & \forall i \\
textpage_{k,i} \rightleftharpoons textpage_{k,i+1} & \forall i \\
module_k \Leftrightarrow video_{k,1} & \forall k \\
module_k \Leftrightarrow soundtrack_{k,1} & \forall k \\
module_k \rightleftharpoons module_{k+1} & \forall k
\end{array}$$

where indexes i, j span over the media instances of module k. The first module is activated by starting the presentation and subsequent modules are activated in sequence by the end of the last video clip of the previous module:

$$\begin{array}{l}
presentation \Leftrightarrow module_1 \\
video_{k,last} \Rightarrow module_{k+1} \ \forall k
\end{array}$$

More care should be put in order to consider the possibility that the voice comments last longer than the movie clips in some cases. In general we cannot know about returned media instances all the properties that would be needed to anticipate their behavior. The relationships holding between modules, video clips and soundtracks (that we have defined *constraints* because are not known until query execution) should in this case specify that the medium which has the longer time span act as a master in activating the other medium:

$$\begin{array}{ll}
max(video_{k,i}, voice_{k,i}) \Leftrightarrow min(video_{k,i}, voice_{k,i}) & \forall i \\
max(video_{k,i}, voice_{k,i}) \Rightarrow max(video_{k,i+1}, voice_{k,i+1}) & \forall i \\
module_k \Leftrightarrow max(video_{k,1}, voice_{k,1}) &
\end{array}$$

where $max(a,b)$ and $min(a,b)$ return the longer and the shorter medium, in terms of time span. They are defined by functions at presentation specification time, and are expanded to specific instances identifiers when actual data are returned.

6 Discussion

A number of issues deserve discussion, due to the many unknown elements with respect to a fully defined multimedia presentation.

Inter-media consistency. We assume that query execution returns sets of results which are consistent by definition. This is a quite obvious assumption, inherited from the traditional database environment where tuples are homogeneous, but in principle it can be false. For example, some dresses could be described only by text pages without movies, or conversely only by movies without related texts. In order to manage these cases we could follow two ways: to extend the *master-slave* relationship between media types to identify relevant instances, not only timing relationhips, e.g.

[4] In the model terminology such groups are called *modules*

by assuming that the movie clip is the master data item, and a text without an associated movie clip is meaningless; or to supply stubs, i.e., empty placeholders for instances which do not have a counterpart in other media types, thus introducing some form of NULL values in multimedia data.

In both cases this issue can be approached by some kind of query post-processing, which should return a consistent and complete set of data, possibly with explicit NULL values. Both solutions are straightforward to implement, and the choice should be related to the desired meaning and appearance of the whole target presentation rather than to abstract considerations.

Boundary constraints. Linking distinct objects into a seamless multimedia composition requires compatible interfaces between components. Differently from textual database reports we cannot guarantee that returned data items can always be integrated in a multimedia presentation by showing the same visual properties. A complete discussion goes beyond the scope of this paper but we can address some main issues here. Given two generic data items, their compatibility is referred to a notion of type equivalence which is related to the set of values they can hold, the operation defined over them, the selection properties that allow to select parts of aggregations, and so on. Multimedia data have a much richer set of properties to be considered. For example, two movies are compatible or not according to a wide spectrum of features: size, color, resolution, frame per second, compression, and so on. Some of these properties are fixed, other can be modified without changing the data meaning. The presence of such differences makes compatibility a matter of substance rather than a matter of pure form.

As an example, the transition between two images is perceived differently not only according the visual properties, but also according to the meaning that the images convey. In a fashion show a sequence of images portraying models with different dresses in the same show room is perceived differently from a sequence of images taken in different rooms. We argue that this problem could be approached as integrity constraints are approached in conventional databases, in order to guarantee formal consistency of sets of related data. To some extent boundary constraints should be defined as a means to measure the consistency of the user perception of the multimedia report. However we feel that a solution to this problem is not close.

Query formulation and integration. We have not discussed issues related to the formulation of queries and to the syntactic and semantic devices needed to relate instances returned from different media repositories, e.g., movie clips, associated voice comments and related pages. This of course is a problem of crucial importance, and we do not claim it is easy to formulate formally and to solve. Automatic correspondence between different types of data cannot be established safely relying only on the interpretation of the data content. Content-based image retrieval systems available today are still far from handling a concept of similarity based on the human perceived meaning of the images. Extending the semantic interpretation to several media adds an unknown amount of complexity.

This problem can be approached with success only if we assume that multimedia reporting shares with textual reporting a correspondence between the syntactic and the semantic level of the query execution. In other words, multimedia data can be integrated by automatic procedures only if some features (tags, metadata, identifiers) can be fully recognized at a syntactic level, and used as semantic indexes to relate instances belonging to different data types. XML is a promising environment for approaching this issues, as its use in some related work demonstrates [15, 16].

7 Conclusion

Multimedia reporting can be viewed as an information retrieval activity which constructs a multimedia presentation integrating the data returned by queries directed to different repositories. The consistency of the presentation requires modelling of the relationships among the media objects which are returned by query execution. The relationships define the compatible execution of media elements in terms of synchronization relationships which drive the overall dynamics of the presentation.

In a previous work we have defined a model for delivering and synchronizing complex multimedia presentations. This model is suited to describe the relationships that must hold among elements of a multimedia report built automatically from results of multimedia database queries. While the spatial and logical composition of the media items returned into a multimedia document has been approached in the literature, the management of the synchronization relationships among the different components has not received too much attention. Our proposal makes a step in this direction, even if further investigation is needed.

References

1. C. Baral, G. Gonzalez, A. Nandigam. SQL+D: extended display capabilities for¡multimedia database queries. *ACM Multimedia Conference.* Bristol, UK, 1998.
2. C. Baral, G. Gonzalez, T. Son. A multimedia display extension to SQL: language design and architecture. *University of Texas El Paso, El Paso*, 1997.
3. J.A. Brotherton, J.R. Bhalodia, G.D. Abowd. Automated Capture, Integration, and Visualization of Multiple Media Streams. *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 54–63, Austin, Texas, June 28–July 1, 1998.
4. A. Celentano, O. Gaggi. A Synchronization Model for Hypermedia Document Navigation. *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 585–591, Como, 2000.
5. A. Celentano, O. Gaggi. Authoring and Navigating Hypermedia Documents on the WWW. *ICME 2001, IEEE International Conference on Multimedia and Expo*, Tokio, August 22-25, 2001.
6. A. Celentano, O. Gaggi. Querying and Browsing Multimedia Presentations. In M. Tucci (ed.), *MDIC 2001, 2nd International Workshop on Multimedia Databases and Image Communication*, LNCS Series 2184, Springer Verlag, 2001.
7. I:F: Cruz, P.S. Mahalley. Temporal Synchronization in Multimedia Presentations. *IEEE International Conference Multimedia Computer Systems (ICMCS '99)*, Vol. 2, pages 851–856, 1999.
8. I.F. Cruz, W.T. Lucas. A Visual Approach to Multimedia Querying and Presentation. *Proceedings of the Fifth ACM International Conference on Multimedia '97*, pages 109–120, Seattle, WA, USA November 9-13, 1997.
9. O. Gaggi, A. Celentano. Modeling Synchronized Hypermedia Documents. *Technical Report n. 1/2001*, Department of Computer Science, Università Ca' Foscari Venezia, Italy, January 2001, submitted for publication.
10. V. Hakkoymaz, J. Kraft, G. Ozsoyoglu. Constraint-based automation of multimedia presentation assembly. *Multimedia Systems*, 7, pages 500–518, 1999.
11. L. Hardman, D.C.A. Bulterman and G. van Rossum. The Amsterdam Hypermedia Model: adding time and context to the Dexter Model. *Comm. of the ACM*, 37(2), pages 50–62, 1994.
12. L. Hardman. Using the Amsterdam Hypermedia Model for Abstracting Presentation Behavior. *Electronic Proceedings of the ACM Workshop on Effective Abstractions in Multimedia.* San Francisco, CA, 4 November 1995.
13. L. Hardman. Modelling and Authoring Hypermedia Documents. *PhD. Thesis*, University of Amsterdam, 1998. `http://www.cwi.nl/ftp/~lynda/thesis`.
14. M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismaïl, L. Tardif. Madeus, an Authoring Environment for Interactive Multimedia Documents. *ACM Multimedia Conference.* Bristol, UK, 1998.
15. J. van Ossenbruggen, F. Cornelissen ,J. Geurts, L. Rutledge, L. Hardman. Cuypers: a semi-automatic hypermedia generation system. CWI Tech. Report INS-R0025, CWI, Amsterdam, 2000.
16. J. van Ossenbruggen, J. Geurts, F. Cornelissen, L. Rutledge, L. Hardman. Towards Second and Third Generation Web-based Multimedia. *Tenth International World Wide Web Conference.* Hong Kong, May 1–5, 2001.
17. L. Rutledge, B. Bailey, J. van Ossenbruggen, L. Hardman, J. Geurts. Generating Presentation Constraints from Rethorical Structure. *Proc. 11th ACM Conference on Hypertext and Hypermedia.* San Antonio, Texas, USA, May 30–June 3, 2000.
18. L. F. G. Soares, R. F. Rodrigues, D. C. Muchaluat Saade. Modelling, authoring and formatting hypermedia documents in the HyperProp system. *Multimedia Systems*, 8(2), 2000.
19. Synchronized Multimedia Working Group of W3C. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. *W3C Recommendation*, 15 June 1998. `http://www.w3.org/TR/REC-smil`.

An End User Retrieval Interface for Structured Multimedia Documents

Andreas Henrich Günter Robbert

Otto-Friedrich University of Bamberg
Department for Databases and Information Retrieval, D-96045 Bamberg, Germany
{Andreas.Henrich|Guenter.Robbert}@sowi.uni-bamberg.de

Abstract

We present an end user retrieval interface for a repository containing large amounts of structured multimedia teaching content. This interface is dedicated to authors of new courses searching for reusable components. Therefore the expressive power of the query opportunities is a dominant design target, but ease of use is nevertheless important. The interface allows to search for arbitrary granules ranging from complete courses to single media objects. It comprises (1) the *query structure window* which allows to define the objects relevant for the query together with their interrelationships, (2) *search detail forms* which allow to define conditions and ranking criteria for the single objects and (3) sophisticated facilities to define the semantics of the combination of different ranking criteria and filter conditions in the context of structured documents.

1 Motivation

The market for computer-based training applications (CBT) and web-based training applications (WBT) is rapidly growing. The objective with respect to the creation of training applications must be to reduce both production costs and production time. Obviously reuse of existing components would be a good step in this direction.

With this scenario in mind, there are two important prerequisites. First, the whole teaching content developed by the organization has to be maintained in a common repository in a structured way. Second, powerful retrieval facilities are needed for this repository to find the desired reusable components in a concrete situation. These retrieval facilities have to be highly expressive in order to define the desired components precisely and they nevertheless have to be easy to use. In our opinion a two tier approach is best suited for this purpose. There should be an application specific graphical user interface for the end-user. This user interface should be implemented on top of an object-oriented general purpose query language comprising powerful facilities to address multimedia data. The retrieval interface accepts queries from the user in a rather intuitive and nevertheless expressive form. These queries are automatically transformed into queries for the underlying object-oriented query language. The basic ideas of the general purpose query language our approach is based on have been presented in [7, 8]. The present paper is dedicated to the graphical end user interface.

To give a first flavor of the user interface, consider a user searching for an image containing a given logo where the text nearby the image is dealing with "data mining and data warehousing". Figure 1 depicts the formulation of this query in our user interface. The structure of the query is defined in the upper right panel (*query structure window*). To this end, the icons from the left panel are placed in the *query structure window* via drag and drop. In the example we see a

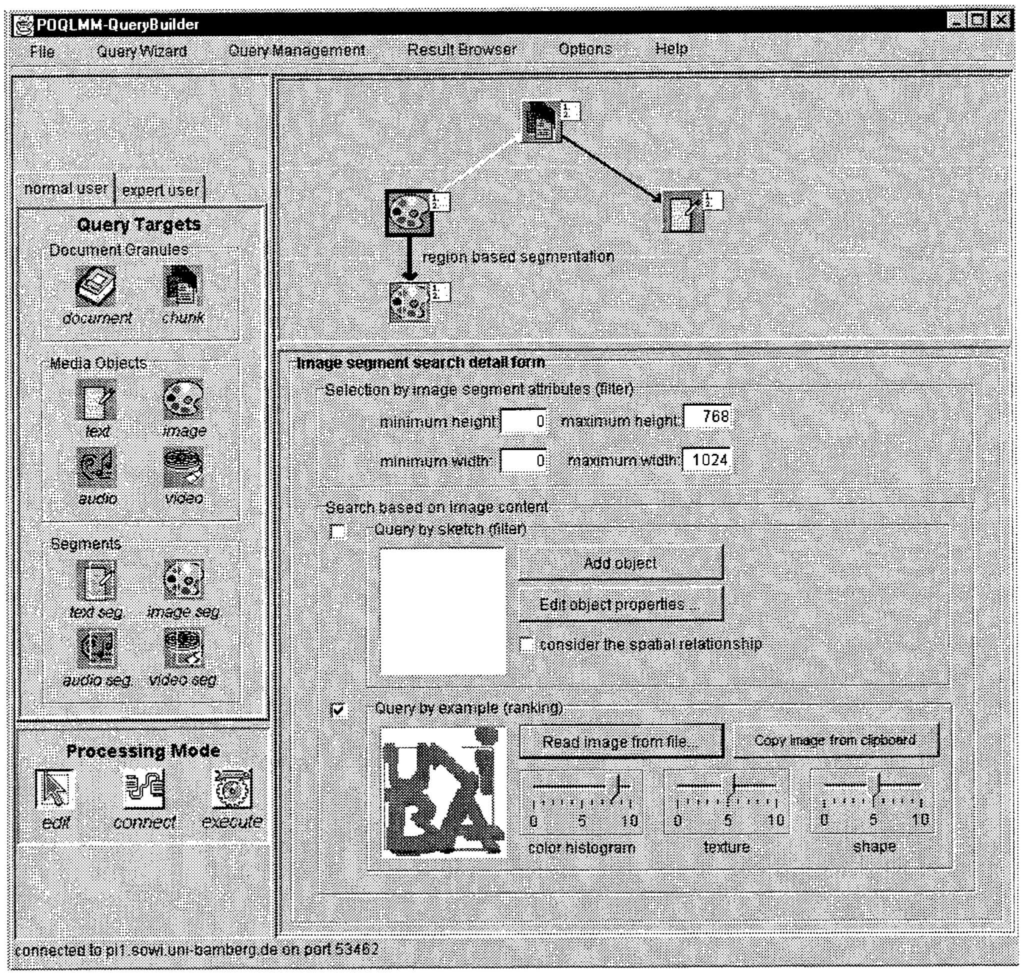


Image segment search detail form

Selection by image segment attributes (filter)

minimum height:	0	maximum height:	768
minimum width:	0	maximum width:	1024

Search based on image content

[ ] Query by sketch (filter)

[ ] consider the spatial relationship

[✓] Query by example (ranking)

connected to pi1.sowi.uni-bamberg.de on port 53462

Figure 1: Example query created with the user interface

chunk object (representing an intermediate document fragment), an *image object*, a *text object* and an *image segment object* in this panel. The corresponding icons are connected to define the desired interrelationships of the objects. In our example the image object emphasized by a black border forms the center of the query. This means that image objects are requested as result. From the corresponding icon a broad arrow is leading down to an image segment icon. This link represents a region based image segmentation. The image segment icon is active (impressed) at the moment. For the active icon the corresponding *search detail form* is shown in the lower right panel where filter conditions and ranking criteria can be defined. In the example a ranking with respect to the similarity to the given logo is requested.

Above the image icon in the *query structure window* is an association to a chunk object representing the part of the document containing the image. From this chunk there is an arrow downward to a text icon representing text objects. In this way text objects occurring in the vicinity of the image object can be addressed. For the text object arbitrary conditions can be defined e.g. regarding the occurrence of terms such as "data mining" or "data warehousing". Roughly spoken the small icon graph in the *query structure window* defines that we are looking for image objects, and that conditions or ranking criteria are defined for associated image segment objects and text objects maintained in the same chunk.

The semantics of the particular elements of the user interface will be described in section 4. Beforehand, we summarize the requirements for the user interface in section 2 and present the example schema presumed in our system in section 3. Finally, some related approaches are considered in section 5.

2 Requirements for the User Interface

For the design of the user interface we have to take into consideration that the typical user is a professional author of multimedia documents. We are not concerned with an unexperienced occasional user of the system. Therefore the focus of the user interface is on its expressive power. In the context of CBT- and WBT-applications, this does especially include the following aspects:

(1) The query user interface (QUI) must allow to search for arbitrary granules ranging from whole documents over intermediate chunks to single media objects. (2) With multimedia data the semantics is usually given implicitly in the media objects, therefore the QUI should allow to *extract features* from the media objects potentially describing their semantics. (3) Because of the vagueness in the interpretation of the media objects and in the expression of the users information need *partial match* and *similarity queries* should be facilitated. (4) Multimedia documents usually contain substantial textual parts. Hence a QUI for multimedia data should admit the use of *text retrieval techniques* and especially the combination of text retrieval techniques with retrieval techniques for other media types. (5) Due to the heterogeneous nature of multimedia applications there is no single combination of different similarity measures fitting well in all application areas. Thus the QUI must facilitate a *flexible combination of different similarity measures* trimmed well for application specific needs.

3 Example Schema

As pointed out in the introduction our approach is based on the assumption that the teaching content is maintained in a common repository. Hence, a schema covering all relevant aspects of the maintained multimedia documents is needed. To this end, we employ the conceptual schema given in figure 2 which is partly deduced from the ideas presented in [1]. It is consciously general, to cover the whole variety of multimedia teaching content. The attribute types applied to each object type are given in the ovals at the upper left corner of the rectangles representing the object types. The relationship types between the object types are indicated by arrows. A double arrowhead at the end of a link indicates that the relationship has cardinality *many.*

The upper right corner of the schema in figure 2 represents the model for the *document structure*. We assume that a multimedia document is made up of one or more substructures called "chunks", which in turn consist of media objects or lower level "chunks". As a consequence *media objects* form the leaves of the tree representing the structure of a multimedia document, and *chunks* represent the internal nodes of this tree. The lower right part of the schema depicts the different types of media objects. There are four subtypes (*image*, *video*, *audio*, and *text*) for the object type *media_object* with specific attributes for registration data. The object type *raw_data* is used to store the raw data of a media object in one or more formats.

The third part of our schema, depicted in the lower left corner, represents the potential segmentation of media objects. For example an image might be segmented into regions containing certain conceptual objects – we call this a *spatial* segmentation – or a video or an audio might be segmented into shots or songs – we call this a *temporal* segmentation.

As suggested in [1] a "multimedia interpretation model" is added to the schema. This part is represented in the upper left corner of the schema. It describes the content of media objects and segments by interpretation objects (*interp_object*). For example an image showing Steffi Graf and Andre Agassi might be associated with two interpretation objects, representing their

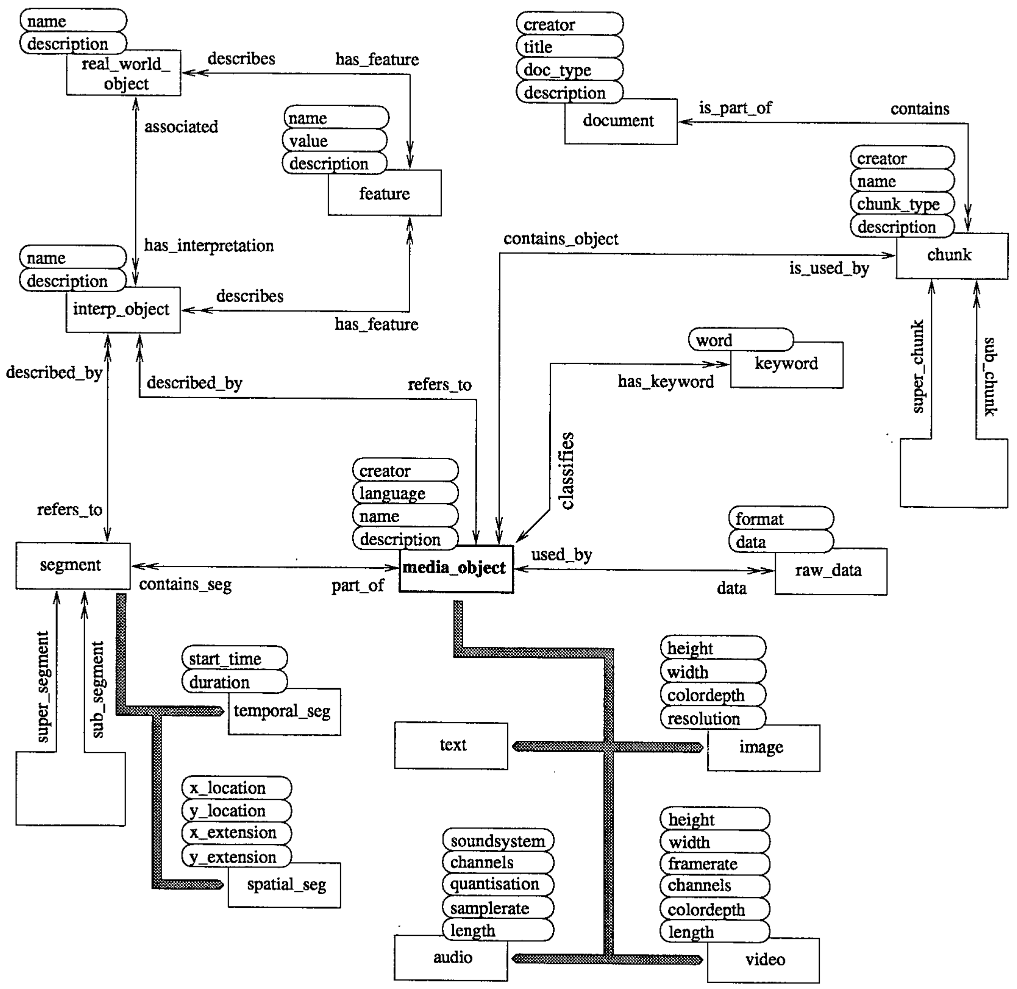


Figure 2: Model for the document structure

appearance in the image. Additionally features of interpretation objects and real world objects can be maintained by the object type *feature*. An example for such a feature would be “mimic”. E.g. there could be a feature object representing that Andre Agassi is smiling in the image. Finally the object type *keyword* allows the assignment of keywords to media objects in order to classify them further.

4 User Interface

The basic idea of the user interface is to define the structure of a query in the *query structure window*. Here “structure of the query” especially means the object types involved in the query and their interrelationships. The desired properties of the single objects are in turn specified via *search detail forms* which exist for all object types (*document*, *chunk*, *image*, *text*, ...). Hence, the query specification in our tool can be subdivided into three subtasks: (1) Specify the concerned parts of the document hierarchy. (2) Specify the conditions for the single query components. (3) Specify the relationships between the query components. In the following we will present the user interface based on this structure.

4.1 Define the concerned parts of the document hierarchy

For all objects, which shall be addressed in the query, corresponding icons have to be dragged from the left panel (Query Targets) into the *query structure window*. The first icon placed in the *query structure window* is peculiar in the sense, that it specifies the desired result object type – i.e. the "central" object type of the query. Therefore this icon is emphasized by a black border.

The user interface provides icons to address *documents*, *chunks* and the four types of media objects, i.e. *text*, *image*, *audio* and *video*. Icons for the corresponding segments of the media object types are available as well.

When placing the icons in the *query structure window* the position of the single elements in the document hierarchy has to be considered. For those icons which have to be connected afterwards the icon representing the components has to be placed in a row underneath the icon representing the superobject.

As an example we resume our query from figure 1. In this query we are searching for images containing a given logo where the text in the vicinity of the image is dealing with "data mining" and "data warehousing". Here an *image* icon is used to represent the desired result object type. The *image segment* icon is placed in the row underneath the image icon because the image segments can be envisaged as components of the image. The *chunk* icon is placed in the row above the image icon because it represents superobjects containing the image under consideration as a component. The *text* icon is placed in the same row with the image icon and also underneath the chunk icon since we want to address text objects which are components of the chunk.

4.2 Conditions for the single query components

To define filter conditions and/or ranking criteria for the objects corresponding to a certain icon, this icon has to be activated by a single mouse-click. Then the *search detail form* associated with the icon type is displayed in the lower right panel of the user interface. Here the desired properties can be specified.

Let us consider the Image segment search detail form displayed in figure 1 as an example. This form is subdivided into two group boxes. In the first group box (Selection by image segment attributes) conditions regarding the size of the segment can be defined. In the second group box (Search based on image content) there are two alternatives to address the image content. The Query by sketch subpanel can be used to define desired properties for the interpretation objects associated with an image segment [1]. To this end, an interpretation object can be added to the small graphic panel via the Add object button. For each interpretation object on this graphic panel conditions based on the *name*, the *description* and associated *feature* values can be defined in a pop-up menu started with Edit object properties . . . . If multiple interpretation objects are considered, the check box consider the spatial relationships can be activated to define the spatial relationships of the objects represented in the small graphic panel as mandatory for the corresponding interpretation objects in the image. The conditions for the interpretation objects associated with an image segment are used as a filter for the image segment objects. In contrast, the requirements defined in the Query by example (QBE) group box are not used as a filter, but as a ranking criterion [4]. Here an image from a file or from the clipboard can be used as an example for the features of the desired image segments. These features include the color distribution, the texture and the shape of the objects. The weights of the single criteria with respect to the overall ranking of the corresponding image segments can be defined using sliders (cf. figure 1). For the actual derivation of rankings for the image segments we use the

[1]This approach is inspired e.g. by the work presented in [13].

ranks of the image segments with respect to the single criteria. Let $r_{i,j}$ be the rank of image segment i with respect to criterion j ($j \in \{1, \ldots, m\}$ and $r_{i,j} \in \{1, 2, \ldots\}$) and let w_j be the weight of criterion j representing its relative importance amongst the criteria. Then we can use the sum $\sum_{j=1}^{m} w_j \cdot \frac{1}{\sqrt{r_{i,j}}}$ to derive the combined ranking. In fact this means that we assign 1 point for the first rank, 0.71 points for the second rank, 0.58 points for the third rank, and so forth. In addition the ranking criteria are weighted with the values w_j as defined via the sliders [2].

Summarizing the semantics of the Image segment search detail form is that the conjunction over all filter conditions is built. For the remaining elements a weighted ranking with respect to the selected ranking criteria is calculated.

The search detail forms for the other segment types and media types are more or less analogous to the described Image segment search detail form. However, one important point is to mention that mature text retrieval techniques are available especially with the Text search detail form. Here pattern matching facilities and a text content based ranking according to the well known vector space model [12] can be used to define filter and ranking conditions.

4.3 Relationships between the query components

In the following we will use the term *query component* to address an icon in the *query structure window* together with the filter conditions and ranking criteria defined for the icon. The semantics of the relationships between the query components covers three main aspects: (1) the actual interrelationship between the objects represented by the *query components* at both ends, (2) the way filter and/or ranking criteria for the destination *query component* are applied to the objects represented by the origin *query component*, and (3) the way multiple relationships originating from one *query component* are combined.

To define the details for a concrete relationship a pop-up menu is opened whenever the user connects two *query components* in the *query structure window*. Figure 3 gives an example for such a dialog which covers the three aspects mentioned above. In the following we will discuss these aspects in more detail.

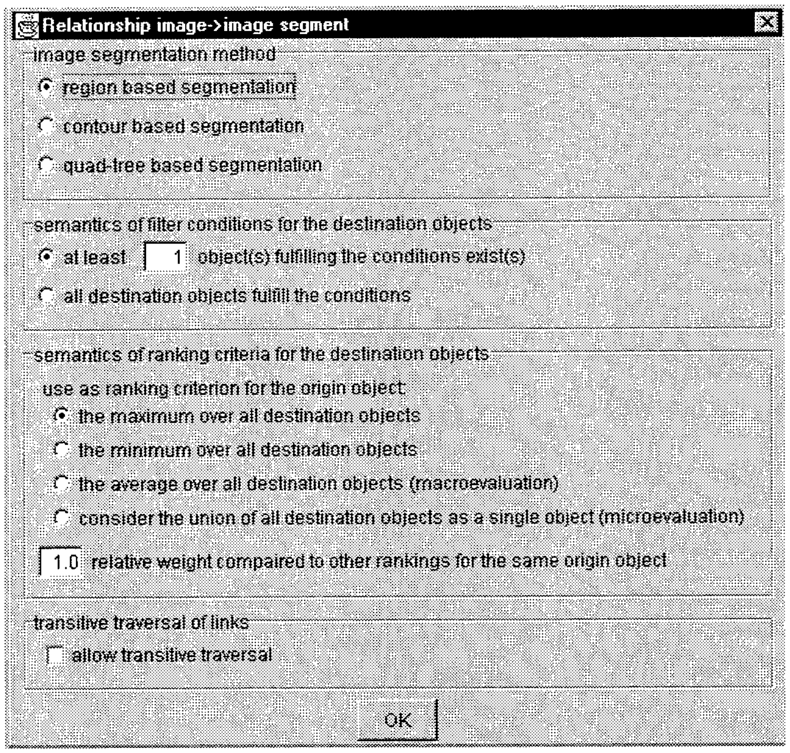
4.3.1 The actual interrelationship between the objects

In principal, three types of relationships between objects can be distinguished: *decomposition*, *segmentation* and *conversion*. An example for a decomposition arises when a *query component* representing chunks is connected with a *query component* representing images. In this case the *query component* for the images represents all image objects, which can be reached from the actual chunk object via a *contains_object* relationship. An example for a segmentation arises when a *query component* representing images is connected with an image segment icon. Finally, an example for a conversion arises when a *query component* representing images is connected with a text icon, meaning, that the text should be derived from the corresponding image via OCR.

For many relationships drawn between *query components* the required actual interrelationship is non-ambiguous. If we connect, for example, a chunk icon with an image icon, this means that there must be a *contains_object* relationship between the objects.

Nevertheless, specific situations are conceivable where details for the actual relationship have to be defined. Assume for example that — in contrast to our example schema — multiple segmentation methods are provided by the system. In this situation the user has to specify the desired segmentation method in the pop-up menu. The group box titled image segmentation method in figure 3 illustrates this alternative. Consequently the pop-up menus for all available

[2]See [9] for a comparison of combination schemes based on ranks vs. concrete similarity values.



Relationship image->image segment

image segmentation method

[✓] region based segmentation

[ ] contour based segmentation

[ ] quad-tree based segmentation

semantics of filter conditions for the destination objects

[✓] at least [1] object(s) fulfilling the conditions exist(s)

[ ] all destination objects fulfill the conditions

semantics of ranking criteria for the destination objects

use as ranking criterion for the origin object:

[✓] the maximum over all destination objects

[ ] the minimum over all destination objects

[ ] the average over all destination objects (macroevaluation)

[ ] consider the union of all destination objects as a single object (microevaluation)

[1.0] relative weight compaired to other rankings for the same origin object

transitive traversal of links

[ ] allow transitive traversal

OK

Figure 3: Pop-up menu for a relationship from an image to an image segment

relationship types have in common that if there is a potential ambiguity in the underlying actual interrelationship, corresponding details have to be specified in the upmost group box of the pop-up menu.

Another aspect of the semantics of the underlying relationship between objects is that transitive traversals may be appropriate in some situations. If we consider our example from figure 1 situations are conceivable where the text objects are not directly connected to the chunk object but indirectly via intermediate chunk objects. In such situations, the check box in the transitive traversal group box has to be marked. In the *query structure window* relationships for which transitive traversal is allowed are represented by dotted lines.

4.3.2 Application of filter and ranking criteria to origin objects

When a *query component* qc_1 is connected with another *query component* qc_2 for which a filter condition and/or a ranking criterion is defined, the question arises, how the filter condition and/or a ranking criterion for qc_2 influences qc_1.

As an example, assume that we are looking for a chunk for which *all* text components are created by "Smith". In this case a chunk icon represents qc_1 and a text icon represents qc_2. For qc_2 the condition *creator* = "Smith" has to be defined in the associated *search detail form*. Furthermore, we have to express the desired "$\forall$-semantics" in the pop-up menu for the relationship from qc_1 to qc_2. This can be done in the second group box (for filter conditions) of the pop-up menu (cf. figure 3). Here the user can select either "exists" together with a minimum required quantity or "for all". "Exists" obviously means that there must be one (or the specified number of) destination object(s) fulfilling the criteria defined for the destination objects of the relationship. "For all" means that all destination objects must fulfill the criteria.

The situation becomes a bit more complicated when a ranking criterion is defined for the destination *query component*. Assume, that we are searching for chunks. We connect the chunk icon with a text icon for which a ranking criterion is defined. In this case, the ranking criterion is defined for the *text* objects but the ranking is in fact needed for the *chunk* objects because the *text* objects are not present in the query result. Therefore a ranking criterion for the chunks has to be derived from the ranking criterion for the text objects. To describe how this ranking can be derived, it is important to remind that the ranking on the text objects in our example is based on similarity values. The same is true with ranking criteria for images and other media types. Now we have to derive a similarity value for the superobject from the set of similarity values of its components or segments. To this end, we can apply the *maximum*, the *minimum* or the *average*. For our example the *maximum* means that we are looking for chunks with at least one "highly relevant" text component. The *minimum* means that we are looking for chunks with no "irrelevant" text components and the *average* is a compromise in between. However, what *average* means finally depends on the applied similarity model. With the vector space model for example we have to take into account that there is usually a document length normalization. This means that components of all sizes have the same influence on the ranking of the chunks. However, it might be more appropriate for an average to consider the concatenation of all text components of the chunk as the critical unit, and to apply the vector space model to this "virtual" text of the chunk. Note that the same principle can be applied e.g. with images where a normalized color histogram is used to derive the similarity values. In this situation it might also be appropriate to consider the union of all pixels in the images instead of simply calculating the average over all similarity values.

As a consequence, the pop-up menu given in figure 3 provides the user with all four alternatives mentioned above.

4.3.3 Combining multiple relationships originating from one icon

In section 4.3.2 we dealt with the semantics of one relationship originating from a *query component*. If there are multiple relationships originating from a *query component*, the interrelationship between them has to be clarified. To this end, we have to distinguish three cases: (1) all destination icons represent filter conditions, (2) all destination icons represent rankings and (3) some destination icons represent filters and some rankings.

Let us start with the first case: multiple filter conditions. An example would be a query where we are looking for chunks containing an image with the name "Sun" and a text from the creator "Smith". The desired semantics of this query can be stated as: We are searching for all chunks c with ($\exists$ image i component of c: *i.name* = "Sun") $\wedge$ ($\exists$ text t component of c: *t.creator* = "Smith"). Therefore an "AND"-semantics is desired for the two conditions. However, we could as well intend an "OR"-semantics, and in situations with three or more connected query components arbitrary Boolean expressions are conceivable. In our user interface, we have solved this as follows: The default combination semantics is "AND". For more complex conditions the expert user mode of the interface provides additional icons with AND, OR and NOT semantics. If the user intends a semantics other than AND, he can employ these icons to specify the desired expression.

Let us now consider the situation where *ranking criteria* are defined for multiple related query components. As an example we can recall our query from figure 1. Here ranking criteria are defined for the *image segments* and for the *text objects in the vicinity*. In this case we calculate the similarity values which shall be applied for the ranking of the *image* objects on the basis of the semantics defined with the relationships originating from the image icon. If we assume that the *maximum* semantics is defined for the image segments and that "micro-evaluation" (cf. figure 3) is defined for the relationships from the image icon to the text icon, this means that four ranking criteria are defined for the images: (1) the similarity of the union over all texts in the vicinity compared to the query text, (2) the color similarity value for the most similar image segment, (3) the texture similarity value for the most similar image segment and (4) the shape similarity value for the most similar image segment. These four similarity values are now combined to a ranking for the image objects analogously to the combination of multiple similarity criteria described in section 4.3.2.

Finally, the semantics for situations where ranking and filter conditions are mixed has to be clarified. Here it is useful to recall that a ranking is a set with an – at least partial – ordering. This means, that we can apply the same mechanisms we applied to combine filter conditions. If we have AND-semantics between a filter and a ranking, only those objects (typically chunks) are considered for which the filter condition is fulfilled and for which there is at least one component to derive the ranking. In other words, we apply the ranking and reject those objects for which the filter condition does not hold. The situation becomes a bit more complex with an OR-semantics. In this case, there are some elements in the result for which no ranking exists. In such situations we define, that the desired objects (typically chunks) for which a ranking exists are ranked ahead of those objects, for which no ranking exists, simply because they do not have corresponding associated objects.

5 Related Approaches

In the recent years graphical user interfaces for information retrieval systems have gained more and more significance. Especially in the range of digital libraries numerous approaches were proposed. Most of these approaches deal with the visualization of search results (an example is Envision presented in [10]).

However, graphical user interfaces are not only useful with respect to the visualization of search results but can also assist users during the creation of queries — as our system does. This applies in particular when queries can address the structure of the documents and the content of multimedia data. An interesting approach in this respect is InfoCrystal [14], a visualization tool and visual query language for boolean and vector space queries. Users can explore an information space along several dimensions simultaneously and manipulate this information by creating abstractions. Arbitrarily complex queries can be constructed by using InfoCrystals as building blocks, organizing them in a hierarchical structure. InfoCrystal enables users to explore and filter text-based documents in a flexible and interactive way, but does not address multimedia aspects. Another interesting approach is PESTO [2] providing querying and browsing of an object database in a hypertext like fashion. The main idea of this approach is a paradigm, called "query in place", that presents querying as a natural extension of browsing. The query facilities of PESTO comprise support for object-oriented queries including path predicates, queries over nested sets, filtered sets and method invocation. But, similar to InfoCrystal, PESTO addresses only the search on textual data.

Contrary to the interfaces specified above, the Delaunay^{MM} system [3] supports an interactive customizable interface for querying multimedia distributed databases. Here users select virtual document styles that cater the display of query results to their needs. Furthermore Delaunay^{MM} provides pre- and post-query refinement and nested queries. Attribute-based search for multimedia data is offered by the Delaunay^{MM} system while content-based search on multimedia data is not possible.

InfoGrid [11] and FireWorks [5] pursue another approach. They offer a framework for building information retrieval applications that support the rapid construction of graphical user interfaces. The InfoGrid framework consists of a set of five classes and object-oriented protocols which build the basis for a default implementation of a user interface. The FireWorks framework is very similar to InfoGrid but offers the possibility to state one query against different repositories. Again, both frameworks only support text-based search, nevertheless our work was inspired by the concepts of FireWorks.

6 Conclusion and Future Work

In this paper we have proposed an end user retrieval interface for structured multimedia documents. One main distinguishing feature of our interface is the support for complex queries on structured documents. This comprises the combination of filter conditions and ranking criteria as well as flexible means to combine different ranking criteria. Future research directions for the interface include the development of a sophisticated query optimizer deriving not only correct but also efficient queries in the underlying query language exploiting e.g. the existing high-dimensional index structures – LSD^h-trees in our case [6].

References

[1] A. Analyti and S. Christodoulakis. Multimedia object modelling and content-based querying. In P. Apers, H. Blanken, and M. Houtsma, editors, *Multimedia Databases in Perspective*, pages 145–179. Springer-Verlag, New York, 1997.

[2] M. J. Carey, L. M. Haas, V. Maganty, and J. H. Williams. Pesto : An integrated query/browser for object databases. In *VLDB'96, Proc. of 22th Intl. Conf. on Very Large Data Bases*, pages 203–214, Mumbai (Bombay), India, Sept. 1996.

[3] I. F. Cruz and K. M. James. A user-centered interface for querying distributed multimedia databases. In *SIGMOD 1999, Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Philadephia, Penn., USA, June 1999.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, Sept. 1995.

[5] D. G. Hendry and D. J. Harper. An architecture for implementing extensible information-seeking environments. In *Proc. of the 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 94–100, Zürich, Switzerland, Aug. 1996.

[6] A. Henrich. The LSD^h-tree: An access structure for feature vectors. In *Proc. of the 14th Intl. Conf. on Data Engineering*, Orlando, FL, USA, Feb. 1998.

[7] A. Henrich and G. Robbert. Combining multimedia retrieval and text retrieval to search structured documents in digital libraries. In *Proc. of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries*, Zürich, Switzerland, Dec. 2000. ERCIM Workshop Reports.

[8] A. Henrich and G. Robbert. POQL^{MM}: A query language for structured multimedia documents. In *Proc. Workshop on Multimedia Data and Document Engineering (MDDE'01)*, July 2001.

[9] J. Lee. Analyses of multiple evidence combination. In *Proc. of the 20th annual Intl. ACM SIGIR Conf. on Research and development in information retrieval*, pages 267–276, July 1997.

[10] L. T. Nowell, R. K. France, D. Hix, L. S. Heath, and E. A. Fox. Visualizing search results: Some alternatives to query-document similarity. In *Proc. of the 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 67–75, Zürich, Switzerland, Aug. 1996.

[11] R. Rao, S. K. Card, H. D. Jellinek, J. D. Mackinlay, and G. G. Robertson. The information grid: A framework for building information retrieval and retrieval-centered applications. In *Proc. of the ACM Symposium on User Interface Software and Technology*, Nov. 1992.

[12] G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, Mass., 1989.

[13] J. Smith and S.-F. Chang. VisualSEEk :A fully automated content-based image query system. In *Proc. of the Fourth ACM Multimedia Conf. (MULTIMEDIA'96)*, pages 87–98, New York, NY, USA, Nov. 1996.

[14] A. Spoerri. Infocrystal: A visual tool for information retrieval & management. In *CIKM 93, Proc. of the Second Intl. Conf. on Information and Knowledge Management, Washington, DC, USA, November 1-5, 1993*, pages 11–20, 1993.

MULTI-MODAL RETRIEVAL FOR MULTIMEDIA DIGITAL LIBRARIES: ISSUES, ARCHITECTURE, AND MECHANISMS

Jun Yang[1], Yueting Zhuang[1] and Qing Li [2]

[1] Department of Computer Science, Zhejiang University, Hangzhou, CHINA
j_yang@yahoo.com; yzhuang@cs.zju.edu.cn

[2] Department of Computer Science, City University of Hong Kong, Tat Chee Ave., KLN, Hong Kong, CHINA
csqli@cityu.edu.hk

ABSTRACT

Supporting effective and efficient retrieval of multimedia data is a challenging problem in building a digital library. In this paper, we examine the issues related to accommodating multi-modal retrieval of multimedia data (text, image, video and audio), and propose *2M2Net* as a generic framework for such versatile retrieval in multimedia digital libraries. The retrieval is conducted based on the integration of multi-modal features including both semantic keywords and media-specific low-level features. This framework is capable of progressive improvement of its retrieval performance, by applying the *learning-from-elements* strategy to propagate keyword annotations, as well as the *query profiling* strategy to facilitate effective retrieval using historic information of the previously processed queries.

1. INTRODUCTION

As the most complex and advanced multimedia information systems, digital libraries are emerging at an increasingly fast rate throughout the world. One of the primary difficulties in building a digital library is to support effective and efficient retrieval of the media objects from the whole library, including text, image, video and audio. The current mainstream of the retrieval technologies in most digital libraries is keyword-based retrieval[5]. Although such technology works well with textual document, it cannot, by itself, accomplish the retrieval task in a multimedia digital library, mainly due to the limited expressive power of keyword to describe or index media objects. Feature-based retrieval, on the other hand, is proposed to index and search for media objects such as image, video and audio [1,2,3] based on their respective low-level features. This paradigm reflects to a certain extent the similarity between media objects at the perceptual level, such as visual and auditory similarity. However, in most cases it cannot achieve the retrieval accuracy that the keyword-based approach can reach, because low-level features cannot be easily associated with the intrinsic semantics of media objects, while keywords explicitly describe the semantics. Therefore, integrating feature-based retrieval with keyword-based approach provides great potentials of improved indexing and retrieval for digital libraries.

No matter which approach is adopted, two problems essential to the retrieval task in the context of digital libraries have to be addressed. First, *how to be multi-modal?* That is, the retrieval approach must be able to search for multimedia data of various modalities (text, image, video and audio), and it should exploit an integration approach to facilitate the multi-modal retrieval task. Second, *how to be progressive?* As a major impediment of retrieval performance, it is commonplace that content of a digital library is not adequately indexed either by semantics or by low-level features. The retrieval facilities must be therefore intelligent enough to improve its performance progressively by learning from the history of previously conducted queries.

To address the aforementioned issues, we propose *2M2Net* as a multi-modal framework for multimedia retrieval in digital libraries. It is characterized as being multi-modal in two aspects:

- Retrieval of various multi-modal data such as text, image and video can be conducted.
- Multi-modal features including both semantic keywords and low-level features are seamlessly integrated for retrieval purpose.

Moreover, this framework employs the following mechanisms to make itself progressive:

- *Learning-from-elements* for propagation of keyword annotation at the semantic level.
- *Query profiling* to facilitate feature-based retrieval based on querying history.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the *2M2Net* framework. In Section 3 and 4, we discuss some key issues regarding semantic-level and feature-level retrieval respectively. We then demonstrate how the feature-level and semantic-level retrieval can be integrated in Section 5. The implementation issues of the prototype system are discussed in Section 6. Finally we present the concluding remarks in Section 7.

2. VERVIEW OF THE 2M2Net FRAMEWORK

The architectural framework of *2M2Net* is illustrated in Figure 1. In the context of this framework, a digital library is viewed as a collection of *multimedia documents*[1], which is recursively defined as a logical document consisting of several elements that are multimedia documents by themselves or individual media objects such as text, image, video and audio. A multimedia document is a semantic grouping of multimedia data, that is, all its elements share a common semantic subject. The concrete forms of a multimedia document can be a web page, a portion of a digital encyclopedia and other forms of multimedia data collection. Since multimedia documents can be constructed recursively, we are able to model many composite documents in real world, e.g. a newspaper, or a website. Multimedia document is internally represented by means of its *semantic skeleton*, which maintains the metadata of both high-level semantics and low-level features for each element in the document.

Multimedia documents are firstly pre-processed so that their various elements are extracted out and stored into the corresponding databases in the **Storage Subsystem**. The metadata of these elements, including semantic keywords as well as media-dependent low-level features, are extracted to constitute the semantic skeleton. User queries are handled by the **Query Processor** at either semantic level or feature level. In the **Feedback & Learning Subsystem**, a set of feedback techniques specific to each media type is utilized for short-term refinement of retrieval results. For long-term improvement of retrieval performance, the *learning-from-element* strategy is applied to propagate and update semantic keywords, and the *query profile* is constructed to facilitate effective retrieval using querying history.

[1] If not indicated explicitly, document is referred to multimedia document in this paper

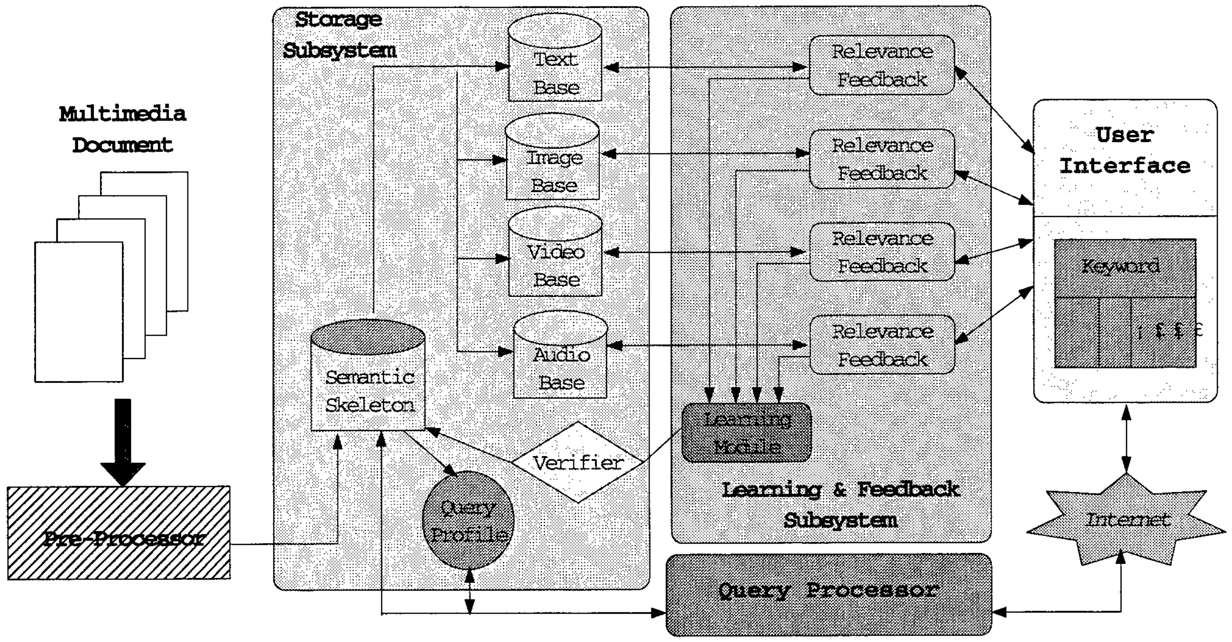


Figure 1: The *2M2Net* framework

3. SEMANTIC-LEVEL RETRIEVAL

In this section, we discuss two key issues regarding this semantic-level retrieval, as initial semantic analysis and the learning-from-element strategy.

3.1. Initial Semantic Analysis

In *2M2Net*, each document and media object has a list of weighted keywords attached to it as their semantic annotation. When a document is pre-processed, we perform semantic analysis to obtain the keyword annotation for the document and its elements. This analysis is straightforward for a textual element, as many traditional IR [5] techniques can be applied to extract representative keywords and calculate keyword weight from itself. The semantics of non-textual object such as image and video, however, cannot be extracted from its content by the current image (video) understanding techniques. Their semantics are acquired indirectly using the following heuristic method.

By our definition of multimedia document, the elements of a document are semantically correlated to each other, so that semantics of a non-textual element can be inferred from related textual elements. Such inference greatly depends on the concrete form of the document. In a digital encyclopedia, the accompanying text and captions of images and videos can be used to represent their semantics. In a Web environment, besides the above text sources, we can take the advantage of using URLs, link strings and other HTML tags to be the descriptions of image and video contained in the web page. The keyword weight is determined heuristically in this case. For example, among all text sources related to an image, we regard the image caption is of the highest relevance and thus give it a large weight. In this way, each keyword can be assigned an estimated weight. The keyword annotation of the whole document can be obtained similarly.

Besides the intra-document correlation used by this heuristic method, there is also inter-document semantic correlation that can be further explored for semantic analysis. Such inter-document correlations widely exist in a digital library, indicated by structural neighborhood and hyperlinks between documents or media objects. By analyzing the structure of these semantic links inside a digital library, we can induce the semantics of a document or a media object from other data directly or indirectly linked to it. However, currently we use solely the intra-document correlation in semantic analysis for the sake of simplicity.

After the keyword annotation become available, the semantic retrieval can be handled by matching the query with the annotation of each candidate document or media object. The matching can be conducted by simply counting the intersection between the query and object annotation in terms of common keywords, or by more sophisticated thesaurus-based semantic similarity measure.

3.2. Learning-from-Elements

The semantics obtained by the heuristic semantic analysis is likely to be incomplete, inaccurate or even non-existing, so that we propose the *learning-from-elements* strategy to propagate and improve the keyword annotations progressively and interactively during relevance feedback. This strategy can be thought as a kind of semantic feedback, because it is triggered when the user submits a set of the documents or media objects as the feedback examples for a given query. As illustrated in Figure 2, it propagate keywords along three directions, which are from user query to feedback examples of documents or media objects (scheme A), from a document to its elements (scheme B) and from a media object to its parent document (scheme C).

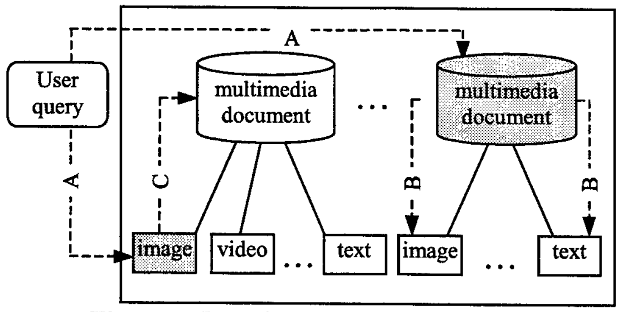


Figure 2: Learning-from-elements strategy

In scheme A, we adopt a simple voting algorithm to update the keyword list of the documents and media objects that are designated as feedback examples. This algorithm is described as follows. For a positive example, we add each query keyword into its keyword list. If the query keyword is already there, its weight is increased by a certain step. For a negative example, if there is any query keyword in its keyword list, we remove it from there. By applying this algorithm on each feedback example, the involved documents and media objects learn their semantics implicitly from users. Besides, the representative keywords with a majority of user consensus are likely to receive a large weight.

Scheme B and C are more ambitious propagation schemes that utilize the intra-document semantic correlation. We apply scheme B if the positive feedback example is a media object. In this case, the keywords inserted or updated (in terms of weight) by scheme A are propagated to its parent document. If the positive example is a document, we use scheme C to spread some of its keywords to its elements. To avoid spreading erroneous keywords, only the keyword with highest weight in the list may be propagated in both scheme B and C, because the top keyword is likely to represent the actual semantics of the document/object. Compared with scheme A, scheme B and C can spread the query keywords to more documents or media objects including those that are not designated as feedback examples, so that they are particularly advantageous when users are reluctant to give many feedbacks. But they are also not as reliable as scheme A. One likely concern of using these two schemes is a tradeoff between wide coverage of keywords and possible erroneous keywords. We argue that a rich set of keywords (perhaps imperfect) is more desirable than a small set of precise keywords for retrieval purpose, and erroneous keywords are relatively easy to identify and correct.

4. FEATURE-LEVEL RETRIEVAL

In this section, we discuss feature-level retrieval as an alternative to the keyword-based retrieval described in the previous section.

4.1. Feature Extraction and Matching

In *2M2Net* framework, the low-level feature of each media object is extracted in the pre-processing phase, such as the color and texture feature for image, structural and motion feature for video, etc. In feature-based retrieval, the user is required to submit a media object as the query example, and the results are retrieved based on the similarity of low-level features. We also incorporate into the framework a set of feedback techniques specific to each media type, including the image feedback technique proposed by Rui [4] and video feedback technique proposed by Wu [6]. Although these techniques are capable of improving retrieval results immediately, they make no contribution to long-term retrieval performance, because they do not memorize the previously conducted feedbacks and start from scratch for the new query.

4.2. Query Profiling

Query profiling is the counterpart of learning-from-elements strategy at the feature level. It can be regarded as an incremental feedback technique that memorizes the history of previous user feedbacks to assist the processing of future queries. The details of this strategy are given below.

For each media object O_i, we construct a query profile to record the objects that were designated as relevant or irrelevant to it in the past feedbacks. The profile is divided into two lists, one for relevant objects (denoted as L_{pi}) and the other for irrelevant objects (denoted as L_{ni}). Each object in the list has a weight attached to it. Initially, the query profile for each media object is empty. Later, when a media object O_i is selected as the query example and some objects are appointed as feedback examples to it, we update its profile using the voting scheme similar to that described in Section 3.2. For each object designated as positive example, we check to see if it is already in L_{pi}. If so, we increase its weight by a certain increment; otherwise we add it into L_{pi} with an initial weight. If we find this object in L_{ni}, it is immediately removed from there. The L_{ni} list of the query example can be updated in a similar way from the negative examples. The query profile can be utilized when O_i is used as query example again. In this case, we select the top *N* objects with the highest weights from L_{pi} and L_{ni} to be the positive and negative feedback examples, respectively. Then, the feedback techniques can be applied directly on these past feedback examples, without any real feedback conducted by the current user.

As more queries and feedbacks are submitted from users, the query profile for each media object can be constructed and improved incrementally. Higher retrieval performance is achievable by using query profiles, since the current retrieval can be brought to a higher level of accuracy achieved by many iterations of relevance feedbacks performed previously.

The query profile described above is associated with each media object and therefore domain-specific. On the other hand, we can construct user-specific profiles that model the different preferences and habits of performing query and feedback for individual users. Finally, personalized query profiles can be established by combing the query profiles and user profiles, which embody the characteristics of both the specific domain and the specific user.

5. INTEGRATION OF SEMANTICS AND FEATURES

In the previous sections, we discussed the retrieval approaches at the semantic level and feature level respectively. In this section, we show that the semantics and low-level features can be seamlessly integrated throughout the whole working flow of the framework to enhance its performance.

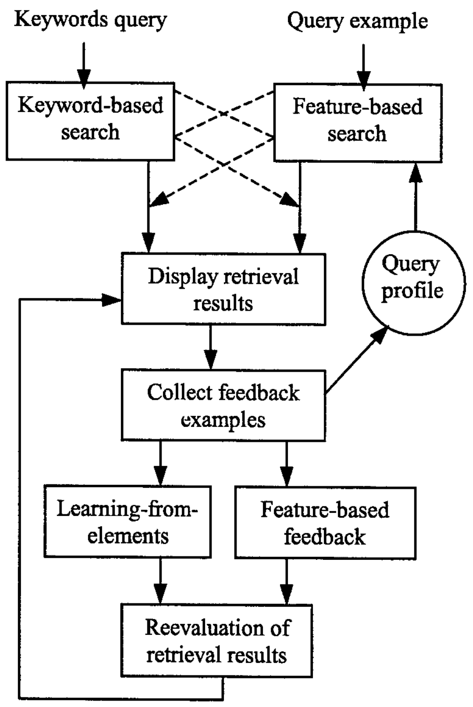


Figure 3: Integration of semantics and low-level features

As illustrated in Figure 3, the user can conduct either a keyword-based search by inputting a set of query keywords, or a feature-based search by submitting a query example. These two search paradigms can be combined to facilitate each other to achieve a better performance. For example, when the keyword annotations of media objects are insufficient, the matches returned by a keyword-based search are usually limited. In this case, we can start a feature-based search using the objects retrieved by keyword search in top ranks as the query examples, in order to find more media objects that resemble them. This second pass search provides many potential results, which, although not very precise, are important supplements to the keyword search. Similarly, feature-based search can benefit from keyword-based approach by accommodating the semantic similarity between the query example and other media objects.

After collecting the feedback examples from the user, the system conducts feedback process in parallel at the semantic level and the feature level. At the semantic level, the learning-from-elements strategy described previously is applied to propagate the query keywords among the related documents or media objects, given that the retrieval is initiated by a keyword-based search. If it is a feature-based search, the keywords in the annotation of the query example are propagated using the same strategy. At the feature level, many conventional relevance feedback techniques are utilized to improve the quality of low-level features, in terms of adjusting their weights or revising the distance metric. The specific technique employed depends on the media type that is currently dealt with, including those for images [4], for videos [6] and possibly for audios (if any). For feature-based queries, we also need to update the query profile of the query example accordingly.

Finally, we recalculate the retrieval results based on the improved semantics and low-level features. Each candidate object is evaluated of its similarity to the query using a comprehensive metric that accommodates its similarity to the initial query, to the positive feedback examples and to negative examples. To achieve this, a uniform distance metric function that measures the similarity between a candidate object O_i and the query is given as follows:

$$S_i = \alpha R_i + \beta \left\{ \frac{1}{N_R} \sum_{k \in O_R} [(1 + R_{ik}) S_{ik}] \right\} - \gamma \left\{ \frac{1}{N_N} \sum_{k \in O_N} [(1 + R_{ik}) S_{ik}] \right\}$$

where α, β and γ are suitable constants, O_R and O_N are set of relevant and irrelevant media objects of a certain type, N_R and N_N are the number of objects in O_R and O_N. R_i is the semantic similarity between the object O_i and the initial query Q, calculated as the number of common keywords between Q and the annotation of O_i. If the retrieval is started with a content-based query, R_i is the similarity between O_i and the object designated as query example R_{ik} is the similarity between O_i and the positive (or negative) feedback example with subscript k, calculated in the say way as R_i in the case of keyword query. S_{ik} is the their similarity in terms of low-level features. For textual object that has no low-level features, S_{ik} is simply set to 1.

6. IMPLEMENTATION ISSUES

A prototype system for multimedia retrieval in a digital encyclopedia has been built based on the proposed framework. The system is composed of a back-end and a front-end. The back-end is responsible for the processing, storage, authoring and retrieval of multimedia data, while the front-end is a web browser based interface that is in charge of all user-system interactions.

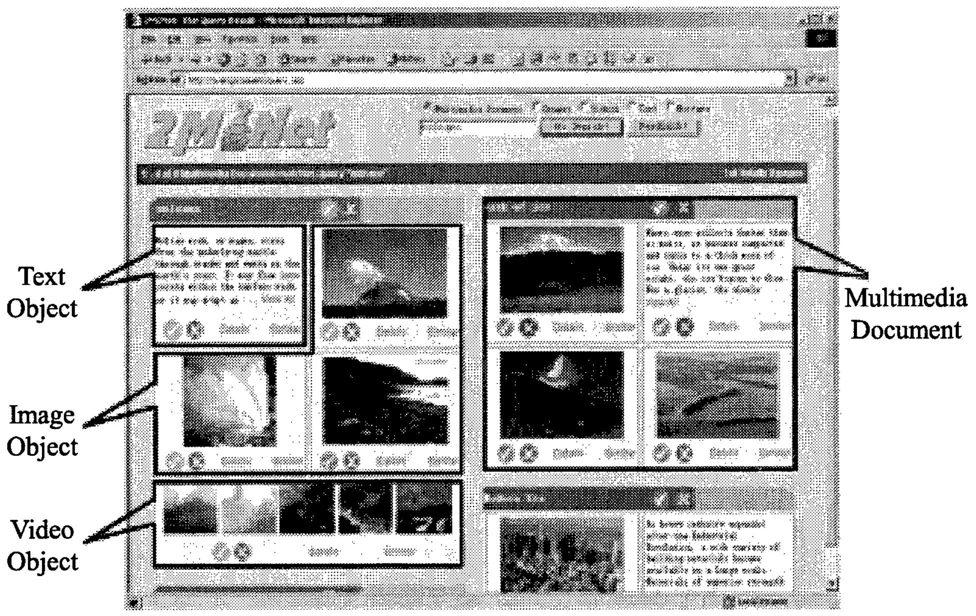


Figure 4: The query results as multimedia documents

The system offers users with great flexibility to perform retrieval task. The user can choose to search for multimedia documents or media objects of a certain modality by a simple keyword-based approach. The main user interface shown in Figure 4 displays the multimedia documents retrieved for the query of "volcano". A multimedia document is rendered as its sketch, i.e. the abstracts of textual objects, thumbnails for image objects and key-frame lists for video objects that are shown together. For each media object, the user can click on the "Details" link below it to view the original media object and other related information. Besides keyword-based search, the user can conduct a feature-based search using a specific media object as the query example by clicking the "Similar" link below it. Each document and media object displayed as retrieval result has a "√" and a "×" icon attached to it that denotes positive and negative example respectively. The user can indicate feedback examples by clicking on the icons and signal the system to perform feedback by clicking the "Feedback" button.

7. CONCLUSIONS

In this paper, we have described *2M2Net* as a multi-modal framework for multimedia retrieval in digital libraries. Among others, *2M2Net* can accommodate retrieval of multi-modal data such as text, image, video and audio, based on the integration of multi-modal features including semantic keywords and the media-specific low-level features. This framework is capable of progressively improving its retrieval performance by applying the learning-from-elements and query profiling strategy. A prototype system has been built upon which the proposed framework is implemented.

Being the most popular operations, browsing and navigation of the multimedia documents are compulsory to be devised. The former is usually facilitated by their subject categories, whereas the latter is to traverse from one document to another related one either semantically or structurally. However, our current prototype system does not maintain any subject category, nor does it track the inter-document links through which the users navigate. In our future work, we plan to include the support of browsing and navigation functionalities into the current prototype system.

REFERENCE

[1] Chang, S. F., Chen, W., Meng, H. J., Sundaram, H., Zhong, D., "VideoQ: An Automated Content Based Video Search System Using Visual Cues", ACM Multimedia, 1997.

[2] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., "Query by image and video content: The QBIC system." *IEEE Computer*, 1995.

[3] Rui, Y., Huang, T. S, Chang, S.F. "Image Retrieval: Current Technologies, Promising Directions and Open Issues", Journal of Visual Communication and Image Representation, Vol. 10, pp39-62, 1999.

[4] Rui, Y., Huang, T.S., Ortega, M., Mehrotra, S., "Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval", IEEE Trans on Circuits and Systems for Video Technology, Special Issue on Segmentation, Description, and Retrieval of Video Content, Vol 8, pp644-655, 1998.

[5] Salton, G., Buckley, C. "Introduction to Modern Information Retrieval", McGraw-Hill Book Company, New York, 1982.

[6] Wu, Y., Zhuang, Y. T., Pan, Y. H., "Relevance Feedback of Video Retrieval", in Proc. of the first IEEE Pacific Rim Conference on Multimedia, pp 206-209, December, 2000.

Querying Images in the DISIMA DBMS*

Vincent Oria[†] M. Tamer Özsu[‡] and Paul J. Iglinski[§]

Abstract

Because digital images are not meaningful by themselves, images are often coupled with some descriptive or qualitative data in an image database. Moreover the division of these data into syntactic (color, shape, texture) and semantic (meaningful real word object or concept) features necessitates novel querying techniques. Most image systems and prototypes have focussed on similarity searches based only on the syntactic features. In the DISIMA system we also propose a solution for similarity searches that combines color histograms, spatial relationships of image blocks and a hash structure to better discriminate among images. Additionally we query images on the basis of salient objects (regions of interest in images) and their properties. This paper presents the querying facilities implemented for the DISIMA system. Both the textual query language (MOQL) and its visual counterpart (VisualMOQL) allow the combination of semantic queries with different types of image queries for better results.

1 Introduction

Multimedia data, especially images, are becoming ubiquitous and are manipulated on a daily basis by computer users. As is the case for all multimedia data, the digital image data does not convey any meaningful information about the image. That is why most image database systems and prototypes focus on content-based image retrieval (CBIR) based visual features such as color, texture and shape [Del99]. The visual features are extracted and represented as points in a multi-dimensional vector space and multidimensional access methods are used to support efficient image searches [SNF00, BBB$^+$97, KSF$^+$96]. The query results returned by these systems can fairly be accurate for some well-defined domains but fail in the general case.

To overcome this problem, an image database has to model and store image semantics that can be used in the querying process. Obtaining these semantics is another issue resolved in most cases using a semi-automated approach. In the DISIMA System, the content of an image is described by means of salient objects (regions of interest) organized hierarchically, following the object-oriented paradigm. The aim of this paper is to show how MOQL [LÖSO97], a declarative query language is used to integrate different image querying approaches: image and salient object semantics, salient object syntactic properties [OÖIL99], and image color distribution [LÖON01].

Section 2 discusses the modelling of semantics in DISIMA. Section 3 presents the querying facilities in DISIMA. Section 4 describes the type system and the query processor. Finally, Section 5 concludes the paper.

*This research is supported by a strategic grant from the Natural Science and Engineering Research Council (NSERC) of Canada and the Institute for Robotics and Intelligent System (IRIS). The work was conducted while the authors were affiliated with University of Alberta

[†]College of Computing Sciences, New Jersey Institute of Technology, oria@homer.njit.edu.

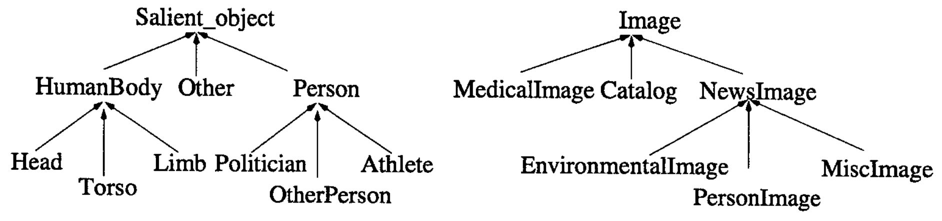
[‡]Department of Computer Science, University of Waterloo, tozsu@db.uwaterloo.ca.

[§]Department of Computer Science, University of Alberta, iglinski@cs.ualberta.ca.

2 Modelling Semantics in DISIMA

Image semantics in the DISIMA system are captured through salient objects, their shapes, and their spatial relationships within images. The DISIMA model [OÖL+97] provides an efficient representation of images and related data to support a wide range of queries. The DISIMA model is composed of two main blocks: the image block and the salient object block. The image block is made up of two layers: the *image* layer and the *image representation* layer. An image is distinguished from its representations to maintain an independence between them.

At the *image* layer, the user defines an image type classification. Figure 1(b) depicts a partial type hierarchy for an application that involves medical images, electronic commerce catalogs, and news images. These first level image types are derived from the type *Image*, the root image type provided by DISIMA. The type *NewsImage* is specialized by three types: *EnvironmentalImage*, *PersonImage*, and *MiscImage*. An image is an object of an image class with some user-defined properties in addition to low-level feature properties such as textures and color distributions (color histograms).



(a) Salient Object Hierarchy **(b) Image Hierarchy**

Figure 1: An Example of an Image Hierarchy.

The *salient object* block is designed to handle salient object organization. A simple example of a salient object hierarchy, corresponding to the image hierarchy defined in Figure 1(b), is given in Figure 1(a). DISIMA distinguishes two kinds of salient objects: logical and physical salient objects (LSO and PSO). An LSO is an abstraction of a salient object that is relevant to some application; it is a meaningful object. A PSO is a syntactic object in a particular image (region of an image) with its semantics given by an associated LSO. A PSO has a shape which is a geometric object stored in its most specific class [OÖIL99], a set of colors and textures.

The object-oriented modelling of geometric objects potentially conflicts with the their mathematical definitions. In [OÖIL99], we provided a more general solution to the shape hierarchy design issue based on the mathematical definitions that allows shape group similarity matches in addition to integrating code reuse at both the data structure and the method levels. The shapes of the salient objects are further used to define some spatial qualitative relations between the objects which are very important in multimedia databases because they implicitly support semantic queries which are captured by qualitative reasoning. The spatial model for DISIMA [OÖIL99] is based on Allen temporal algebra [All83]. The shape of the objects are projected onto the axes and the intervals are combined with the allen's temporal operators to define some topological relations in one-dimensional space. The one-dimensional relationships are further combined to define the spatial relationships.

DISIMA permits feature-based search in addition to search based on image content semantics. This allows more sophisticated queries such as the following: (Find images of flowers taken by John that look like this one). The fundamental elements of histogram-based image retrieval include the selection of the color space, the color space quantization, and the histogram distance metric. There is no general agreement as to the most suitable color space for color histogram-based image retrieval. This is a result of the fact that color perception is highly subjective. Therefore, a variety of color spaces are used in practice such as RGB, HIS, or L*u*v*. In our work, we have chosen to represent the color space using the RGB model. Besides extracting color histograms from entire images, an image can also be segmented into several blocks, each of which has an associated color histogram. These color histograms together form *multi-scale color histograms* of an image. The multi-scale color histograms are used to define image multi-precision similarities [LÖON01].

3 Querying Images

Querying images on salient objects and their properties assumes the detection of these objects. The image annotation is semi-automatic. The image syntactic features are automatically extracted and a human-annotator adds the semantic information on the salient objects in the image. In addition, an image has some descriptive properties (i.e., meta-data), such as date and photographer, that have to be provided. Till now, multimedia data are produced without accompanying meta-data and a human-annotator is often solicited to get this information. This is changing. The emergence of MPEG-7 [Gro01] will lead to media production together with the description of the content descriptions that will provide information such as salient-objects. For the rest of the paper, we assume that the information on salient objects is provided.

3.1 Querying Semantics with MOQL

MOQL (Multimedia Object Query Language) is a text-based multimedia query language [LÖSO97], which is an extension of the standard OQL language [CBB$^+$97].

Most extensions introduced to OQL by MOQL are in the **where** clause, in the form of four new predicate expressions: *spatial_expression*, *temporal_expression*, *contains_predicate*, and *similarity_expression.* The *spatial_expression* is a spatial extension which includes spatial objects, spatial functions, and spatial predicates. The *temporal_expression* deals with temporal objects, functions, and predicates for videos. The *contains_predicate* is defined as: *contains_predicate* ::= *media_object* **contains** *salientObject* where, *media_object* represents an instance of a particular medium type, e.g., an image or video object, while *salientObject* is an object within the *media_object* that is deemed interesting (salient) to the application (e.g., a person, a car or a house in an image). The **contains** predicate checks whether or not a salient object is in a particular media object. The **similarity** predicate checks if two media objects are similar with respect to some metric.

3.1.1 Querying Images Through Salient Objects

The following are two examples of queries expressed in MOQL. The first query looks for images with people and the second finds all image in which a politician named "Chretien" appears.

Query 1 Find all images in which a person appears.

```
select  m
from    Images m, Persons p
where   m contains p
```

Query 2 Find all images in which a politician named Chretien appears.

```
select  m
from    Images m, politician p
where   m contains p
And     p.lastName = "Chretien"
```

Queries in MOQL can easily become non-trivial to express. A query Q: "Find images with 2 people next to each other without any building, or images with buildings without people" can be expressed in MOQL as follows:

Query 3 Find images with 2 people next to each other without any building, or images with buildings without people.

```
select  m
from    image m, building b1, person p1, person p2
where   ( m contains p1 and m contains p2
        And     p1.MBB west p2.MBB
        And     m not in
                (Select m1
                From image m1, building b2
                Where m1 contains b2))
Or      ( m contains b1
        And     m not in
                (Select m2
                From image m2, person p3
                Where m2 contains p3))
```

This example points to the need for a visual query interface. Although the user may have a clear idea of the kind of images he/she is interested in, the expression of the query is not straightforward. The logic in VisualMOQL is based on the observation that queries expressed in natural languages are often composite. VisualMOQL provides a way to construct complex queries by composing simple query blocs or sub-queries.

3.2 Querying Semantics with VisualMOQL

VisualMOQL provides an easier way to express queries, and then translates them into MOQL. Query Q can be decomposed into 2 sub-queries Q_1: "Find images with 2 people next to each other without any building" and Q_2 " Find images with buildings without people". Each of these sub-queries can further be decomposed into 2 simpler sub-queries.

User can choose the image class they want to query and the salient objects they want to see in the images. They can also specify the maximum number of images they want, and the similarity threshold (for similarity queries). The working canvas is where users construct simple queries. They can insert the salient objects that they want to see in images, into the working canvas. The spatial relationships may also be specified between the salient objects. The color, texture, and shape properties of images and salient objects can be specified through a dialog box. After users finish constructing a simple query in the working canvas, it is moved into the query canvas. Several simple queries are combined in the query canvas to form a compound query. Finally, the user presses the query button to submit the query. The VisualMOQL query specified in the query canvas will then be translated into an MOQL query before being submitted to the query processor.

3.3 Feature-Based Searches in DISIMA

DISIMA permits feature-based search in addition to search based on image content semantics. This allows more sophisticated queries such as the following: (Find images of flowers that look like this one).

3.3.1 Color and Texture Similarity for Salient Objects

Since colors and texture are perceived differently by people, searching for an exact match, even at the salient object level, will yield poor results. Color and texture comparisons are done consequently through similarity searches. The user can give the RGB values for the color if he/she knows it or pick a color from an appropriate window to get the color value. The same thing applies to textures. For example:

Query 4 Find all images that contain a salient object with a color similar at 80% to the RGB value (255,0,255) and similar at 70% to the texture value (0.6).

```
select   m
from     Images m, LSO o
where    m contains o
And      o.color similar colorgroup(255,0,255) similarity 0.8
And      o.texture similar texturegroup(0.6) similarity 0.7
```

3.3.2 Shape Similarity

The *Geometric_Object* class supports three types of similarity match: *full-group, class,* and *sub-group,* depending on the similarity threshold specified in the query. The *ellipse* group includes the *Ellipse* and *Circle* classes. The *polyline* group includes the *Polyline* and *Segment* classes. The *Polygon, Rectangle, Square,* and *Triangle* classes belong to the *polygon* group. The shape similarity algorithm we used is the *turning angle algorithm* [ACH+91] because it is orientation invariant. A shape similarity query can be posed with or without a given shape as illustrated by the two following examples:

Query 5 Find all images containing salient objects with a rectangular shape.

```
select   m
from     Images m, LSO o
where    m contains o
And      o.shape similar rectangle similarity 0.5
```

When a shape is not given (i.e., only the shape class is given) in a shape similarity query, the query is processed without any similarity metric. For the example, the query processor will select images for which at least one salient object has a shape of interface type *Rectangle.* The question is which extent to use (shallow or deep extent)? This decision is made with regard to the similarity threshold in the query. If the similarity threshold is set to 1, the query is processed using the shallow extent (class match) otherwise the deep extent is used (sub-group match).

Query 6 find all images containing salient objects with a a shape 50% similar to the rectangle((1,2),(10,2),(10,7)).

```
select   m
from     Images m, LSO o
where    m contains o
And      o.shape similar rectangle((1,2),(10,2),(10,7)) similarity 0.5
```

If we let ϵ denote the threshold, r the rectangle ((1,2),(10,2),(10,7)), S a set of shapes, PSO a set of physical salient objects , and I a set of images, then the solution of the similarity query is $\{i \in I | \exists s \in S, \exists o \in PSO, d(s,r) \leq \epsilon \wedge s = shape(o) \wedge i \ contains \ o\}$ where d is a distance function using the shape similarity algorithm. The problem here is to find the minimal extent that contains all the shapes satisfying the conditions. For example, if we are trying to match a given rectangle within a similarity threshold of 0.9, should we begin our search with the deep extent of all polygons or simply confine our search to the extent of all rectangles? Are we willing at higher thresholds, to miss matching some polygons when we match against a similar rectangle? Of course the lower ϵ is, the wider both the solution and the shape search space will be. In the current implementation, full-group match is applied when the similarity threshold in the search condition is less than 1. Class match is applied when the similarity threshold equals 1 (exact match).

3.4 Image Similarity Queries Using MOQL

The *similarity_expression* can also be used to check whether two images are similar with respect to the metric defined in the multi-precision similarity algorithm.

3.4.1 Image Similarity Queries

For querying images by color histogram matching, two kinds of *similarity_expression* are used to check if two images are similar. One is whole-image similarity queries. For example:

Query 7 Find images that are similar to the user-provided image e1, with respect to color histogram matching at precision level 1, with the similarity threshold 0.8.

```
select   m
from     Images m
where    m.color_histogram similar e1.color_histogram
         precision 1 similarity 0.8
```

The other kind of expression is querying sub-images. For example:

Query 8 Find images whose left halves are similar to the user-provided image i, with respect to color histogram matching with the similarity threshold 0.6.

```
select   m
from     Images m
where    m.color_histogram similar i.color_histogram
         quadrants (1, 2) similarity 0.6
```

3.4.2 Combining Similarity Queries with Semantics

A similarity query can be combined with some semantic properties for a more precise response. The result of such a query is all the images similar with respect to the similarity metric which also satisfy the semantic conditions. For example:

Query 9 Find images with people that are similar to the user-provided image e1, with respect to color histogram matching at precision level 2, with the similarity threshold 0.8.

```
select   m
from     Images m, person p
where    m contains p
And      m.color_histogram similar e1.color_histogram
         precision 2 similarity 0.8
```

The same applies to sub-image queries:

Query 10 Find images with people whose left halves are similar to the user-provided image i, with respect to color histogram matching with the similarity threshold 0.6.

```
select   m
from     Images m, person p
where    m contains p
And      m.color_histogram similar i.color_histogram
         quadrants (1, 2) similarity 0.6
```

Note that the precision levels are not defined for the sub-image queries; that is, all the sub-image queries are carried out at the first precision level.

3.5 Image Similarity Queries Using VisualMOQL

An image in the result of a query can be selected as an entry to a similarity query. A dialog box is brought up by pressing the image property button under the working canvas. The right part of the dialog box is for textual properties such as title, publisher, creation date, etc. The left part of this dialog box is for color histogram similarity matching. The similarity can be done on the whole image or a sub-image. For sub-image queries, users can select the region that they want to query on. However, the size and position of the region is limited by the grid partitions that the system provides [LÖON01].

4 Type System and Query Processor

The DISIMA prototype is implemented on top of a commercial object database. The type system provides the data structures to store and index the images that are used by the query processor we implemented to answer user queries.

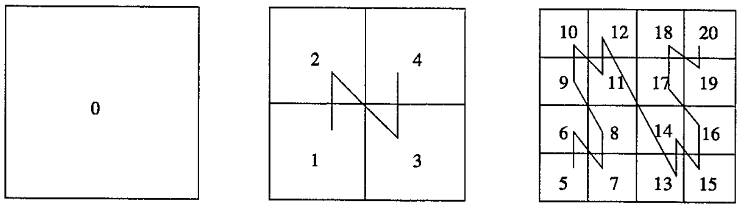
4.1 Type System

The DISIMA type system has been extended to include some structures needed by the multi-precision similarity algorithm. The DISIMA type system is the implementation of the DISIMA model. It provides a root type (or class) for each layer of the model: *Image, Image_Representation, LSO, PSO* and *PSO_Representation.* By means of schema specifications, *Image* and *LSO* are subtyped by the application developer to define application-specific types. *PSO_Representation* has two subclasses: *Raster_Representation*, which is similar to *Image_Representation*, and *Vector_Representation*, which represents the geometry of physical salient objects. Figure 3 shows a high level view of the classes used in the DISIMA type system. The Image, Image Representation, PSO, and LSO classes have been introduced before. The MBB (Minimum Bounding Box) class defines the spatial feature; the Geometric Object class defines the shape feature; the Texturegroup class defines the texture feature; the Colorgroup class and the Multi-scale Color Histogram class define the color feature. The straight line connecting two classes represents the relationship between the classes. The numbers on the straight lines indicate the cardinality of the relationship. The Color Histogram class stores the multi-scale color histograms as quadtrees (Figure 2). An integrated indexing structure (3DEH) is used to index average colors of these color histograms to facilitate image/sub-image queries by color histogram matching [LÖON01].

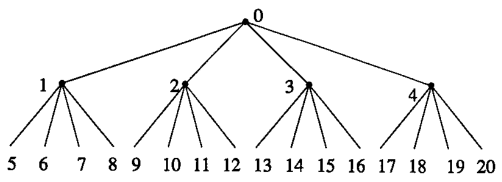
4.2 Implementation of the Index Structure for Image Similarity

While tree-based structures are good for supporting nearest neighbor search, there is considerable traversal that needs to be done. In the DISIMA system, we implemented a hash structure for multi-dimensional indexing called three-dimensional extendible hashing (3DEH). Like traditional hashing, this structure provides efficient, direct addressing of the targeted buckets.

The hash directory of the three-dimensional extendible hashing has three *initial depths* (d_1, d_2, d_3), one for each of the R, G, B color components; it also has a *growth depth* d_g, which is 0 at the



(a) N order of image blocks



(b) quadtree with predetermined children node order

Figure 2: A quadtree stores color histograms of image blocks

beginning and will increase as the address space increases. The number of bits of a hash address is $(d_1 + d_2 + d_3 + d_g)$, so the hash directory has $2^{(d_1+d_2+d_3+d_g)}$ entries. The bucket in three-dimensional hashing has three local depths (p_1, p_2, p_3), which means that all records in the bucket have common p_1, p_2, p_3 leading bits of the R, G, B values, respectively. At the beginning, the local depths of buckets are the same as the initial depths of the directory. The directory entry either points to a bucket or holds a null value if no data records are hashed to this entry.

When a bucket overflows in three-dimensional hashing, like traditional extendible hashing, the hash address space increases and the bucket splits. Unlike traditional extendible hashing, in which a bucket can be split along only one dimension, a bucket in three-dimensional extendible hashing can be split along any of the R, G, B dimensions. We split the bucket along the dimension with the highest variance so that the records can distribute as evenly as possible in the two resulting buckets.

Since the bucket can be split along any one of the three dimensions, we need to record in which dimension it is split. A data structure named *mask track* is maintained to keep track of the splitting history of the bucket. For example, if the bucket 010 split along the R dimension, we record the fact that no buckets have been split, except that the bucket with the initial hash address 010 is split along R dimension in the mask track (Figure 4). That is, every entry in the mask track is zero, except that the 010 entry is 100. We use 3 bits to record that information but the space used for this purpose can be optimized by using fewer bits (e.g., 2 bits for three dimensions). The experiments ran, reported in [LÖON01], show that the 3DEH significantly out performs the SR-tree [KS97]

The three-dimensional extendible hashing is designed to index average colors of the images and their quadrants. If the images are not categorized in the database, then all the images will be simply inserted into one index structure. However, the images in the DISIMA system are organized as hierarchical image classes, so the index structure has to correspond to the hierarchy of images. That is an index for each image class,

4.3 Query Processing

Although ObjectStore provides some querying facilities over collections, it does not have a built-in declarative query language. Therefore, we have fully implemented a MOQL parser and query processor for MOQL queries. The result of the parser is an internal query tree structure which is later transformed

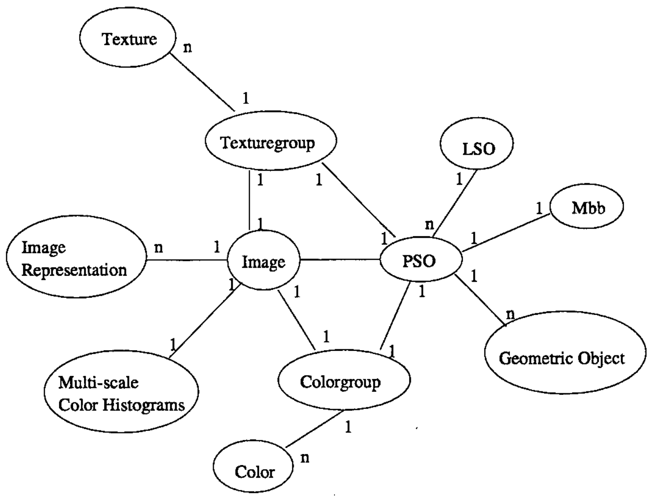


Figure 3: The DISIMA type system overview.

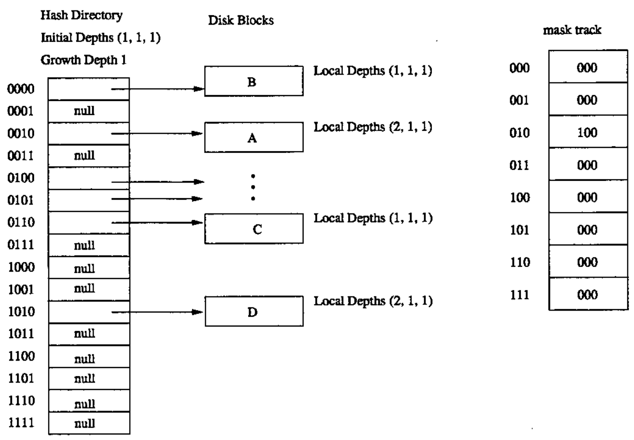


Figure 4: First expansion of the hash directory

into an execution plan.

As MOQL queries follow the same SELECT-FROM-WHERE structure as traditional queries, the design of the DISIMA parser is able to make use of basic rules defined in SQL parsers. The new rules defined on top of the basic rules deal with objects and the clauses introduced by MOQL. The objective of the DISIMA parser is to check the semantics and syntax of the external query, which is in the form of a character string. The parsed string is then converted into a query tree. A query object stores all the information given by the query string.

The query engine uses the query tree directly to generate a non-necessarily optimized execution plan. In the query tree, each node is associated with either one or two conditions. When there are two conditions, either intersection or union is performed upon the results of the conditions for *and* and *or* operators, respectively. Once the query tree is constructed, the query engine initiates post-order traversal. The left-condition is executed before the right-condition, and any sub-node before the higher-level node.

5 Conclusion

In this paper, we have presented the querying functionality of the DISIMA DBMS. We have shown how the the integration of different image query types was achieved in the DISIMA syatem through a declarative query language and a rich type system. We have extended both MOQL and VisualMOQL to support image similarity. Hence, an image similarity search can start with a semantic search to reduce the query domain before the similarity matches. In the current implementation, the semantic information is provided by a human-annotator. But the emergence of MPEG-7 will facilitate the production of media objects together with their content descriptions.

References

[ACH+91] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3), March 1991.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832—843, 1983.

[BBB+97] S. Berchtoll, C. Bohm, B. Braunmuller, D. Kein, and H. Kriegel. Fast parallel similarity search in multimedia databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 1997.

[CBB+97] R. G. G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, 1997.

[Del99] A. Del Bimbo. *Visual Information Retrieval.* Morgan Kaufmann Publishers, 1999.

[Gro01] MPEG-7 Working Group. The MPEG-7 web page. http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm, 2001.

[KS97] N. Katayama and S. Satoh. The SR-tree: An index structure for high dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 369—380, Tucson, Arizona, May 1997.

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Propopapas. Fast nearest neighbor search in medical image. In *Proceedings of the 22nd International Conference on Very Large Databases, VLDB*, Bombay, India, September 1996.

[LÖON01] S. Lin, M. T. Özsu, V. Oria, and R. Ng. An extendible hash for multi-precision similarity querying of image databases. In *Proceedings of the 27th VLDB conference*, Rome, Italy, September 2001.

[LÖSO97] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, pages 19—28, Como, Italy, September 1997.

[OÖIL99] V. Oria, M. T. Özsu, P.J. Iglinski, and Y. Leontiev. Modeling shapes in an image database system. In *Proceedings of the 5th International Workshop on Multimedia Information System*, pages 34—40, Indian Wells, California, October 1999.

[OÖL+97] V. Oria, M. T. Özsu, X. Li, L. Liu, J. Li, Y. Niu, and P. J. Iglinski. Modeling images for content-based queries: The DISIMA approach. In *Proceedings of 2nd International Conference of Visual Information Systems*, pages 339—346, San Diego, California, December 1997.

[SNF00] R. O. Stehling, M. A. Nascimento, and A. X Falcao. On "shapes" of colors for content-based image retrieval. In *Proceedings of ACM Multimedia 2000 Workshops*, pages 171—174, Los Angeles, California, November 2000.

PAMIR: A Parallel Multimedia Information Retrieval System

Adil Alpkocak, Tuba Ulker, Taner Danisman

Dokuz Eylul University
Department of Computer Engineering
35100 Bornova, Izmir, TURKEY
{alpkocak, tuba, taner}@cs.deu.edu.tr

Abstract. *This paper describes the design, implementation and evaluation of Parallel Multimedia Information Retrieval system. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both insertion and processing phase of complex similarity queries in Multimedia Information Retrieval. The system is capable to index and query multimedia objects concurrently based on more than one features (i.e., global color, local color, texture, shape) and each feature of an object queried by a different workstation using multithreaded version of M-tree.*

The system was tested for a collection of approximately 68000 images and each image described by three features; color moments, histogram layouts and texture. Experimentation has been shown that the system provides a good reduction in CPU times on both insertion and query processing phase. Detailed explanation of the system and experimentation results also presented.

1 Introduction

State-of-the art in content-based multimedia retrieval involves two main issues: f*eature extraction*, which extracts feature with distinguishing power from the original Multimedia Objects (MOB), and *feature indexing*, which organizes the objects in the database according to their extracted features in such a way that at run time, only a small portion of the database needs to be explored to retrieve the target objects. The features are extracted directly from the Multimedia Objects (MOB) with as less human intervention as possible. Feature extraction is a very old research area and the literature involves many of the well-known techniques to extract meaningful features from various types of signals such as images, video and audio. In content-based multimedia retrieval, the most common features for MOBs are shape, texture, color for still images, loudness and harmonicity for sounds, shots and objects' trajectories for video etc. Although these features are very common and successfully implemented by many content-based information retrieval systems, the feature extraction phase alone is still a CPU bound problem, particularly, when audio or video is the case.

In a Multimedia Information Retrieval System, the extracted features are then mapped into points in *d*-dimensional vector space and queries are processed against those features vectors. Therefore, a multimedia database can be envisioned as a collection of feature vectors representing MOBs. The Similarity of two images is defined as a distance of their representing feature vectors in *d*-dimensional vector space. In image databases, a typical similarity query corresponds to a nearest-neighbor or range query in *d*-dimensional feature space.

To date, many of the similarity indexing structures has been proposed such as TV-trees, R-trees, SS-trees, X-trees and M-tree. But, use of a similarity indexing is not generally fulfills the constraints about similarity searches. This is because, similarity indexing is both CPU and I/O bound due to the fact that the distance computations are CPU intensive tasks and image data can be too large. Moreover, contents of a MOB may have different properties, i.e., different dimensionality, and distance/similarity metrics. In other words, in a multimedia repository, each MOB is represented by a set of features vectors that may have different properties. Therefore, a multimedia retrieval system should be able to index every features used according to their properties.

On the other hand, content-based multimedia queries do not have to be based on one single feature, and the queries are generally based on the similarity of objects using several feature attributes like shape, texture, color or text. These types of multi-feature queries are called *complex similarity queries* consisting of more than one similarity predicate. Complex queries return a ranked result set with a calculated cumulative score. The results are presented to the user in rank order with highest scores indicating a better match. Within this context, a query based on only one feature is called as *atomic* query and a complex query is considered as a combination of atomic queries.

Some recent work has been done on processing complex similarity queries. In [CPZ97], the authors proposed a solution by extending M-tree indexing structure. However, they have concentrated on a situation where all similarity predicates refer to a single feature. In [CP00], another extension was proposed and original M-tree was renamed as M^2-tree, to perform complex similarity search over multimedia objects represented by multiple features. Both study focused on *feature indexing* issue of content-based multimedia retrieval.

In [Fag99], Fagin proposed a solution that the k best matches are returned for a complex query carried out by independent sub-systems, and that access to any one system is synchronized with the others. This algorithm is called A_0. It has problems of needing too many random accesses. A_0 algorithm formed a base for new researchers. In [NR99], another algorithm, based on Fagin's algorithm, was proposed and called a multi-step query processing algorithm. They claimed that it performs better than Fagin's algorithm by reducing number of database touches on random access phase. In [GBK00], the authors has modified Fagin's algorithm again, and called, quick-combine and proved that it was more efficient by a factor of 30. In [GBK01], authors presented a new algorithm called stream-combine, that allows retrieving the *k* top-scored objects from a set of different ranked result list without needing any random accesses where the order of the objects in resulting *k* set is not important.

Based on these observations, we present the parallel multimedia information retrieval (PAMIR) system, which is designed to address issues for both in *feature extraction* and *feature indexing*. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both search and insertion process of multimedia retrieval. The architecture of PAMIR involves a dedicated workstation for each features of MOB. Each workstation is responsible for both extracting respective features directly from MOB in insertion phase, and indexing those extracted features using modified multi-threaded version of M-tree. Although the main idea behind the PAMIR is quite simple it provides a good speed up in both database population and query processing response time, and it is scalable with the increasing number of features used in the system.

The remainder of the paper is organized as follows. Section 2 describes the problem, presents the architectural characteristics of PAMIR and explains the basic operations such as object insertion and query processing. In Section 3, we present the results obtained from the performance evaluation experimentations. Section 4 concludes the paper and provides an outlook for our future work on this subject.

2 Parallel Multimedia Information Retrieval Environment

A Multimedia Information retrieval system locates and retrieves objects that are relevant to user queries from database. A multimedia object can be an article, reports, image, chart, audio or video. So, multimedia information retrieval systems are designed with the objective of providing, in response to a user query, references to items that contain the information desired by the user.

Retrieving relevant multimedia objects from a large collection is of great importance in almost all types of multimedia information. Large databases create a performance problem and complex feature description consumes too much CPU time. While solving this problem, the throughput of the system must be increased and the response time of the individual queries must be decreased. In order to fulfill these requirements we believe that parallelism may help to this problem, therefore, we propose the PAMIR – Parallel Multimedia Information Retrieval system.

2.1 Basic Definitions

To clarify the later details, let us consider a multimedia system containing the set of multimedia objects, $MMS = \{M_1, M_2, ..., M_m\}$, where each multimedia object is defined by its feature vectors, $M_i = \{F_1, F_2, ..., F_n\}$,where $1 \leq i \leq m$, and a feature vector is defined by its properties $F_i = <f_{i1}, ..., f_{ip}>$, where $1 \leq i \leq n$ and each F_i may have different dimension. In addition, let a set of similarity (distance) functions, $SIM=\{ SIM_1, ..., SIM_n\}$, be defined to calculate spatial proximity between Fs, where SIMs may not be identical.

A multimedia query Q can be formulated using these feature vectors as follows:

$$Q = \{F_1, F_2, ..., F_n\}$$

In a system like this, the query processor requires an organization, which facilities locating relevant objects with respect to their distinct features.

The main issue for parallel system is how we distribute all the multimedia information in the whole collection to the processors in such a way so that all advantages of parallelism could be handled. So, the problem is how to partition the data? There may be two methods for partitioning the data over the processors. These are *horizontal partitioning* and *vertical partitioning*.

To clarify how partitioning handled, let us consider an example involving an image database where each image is described by three feature vectors: color histogram, texture and shape information of the image. More formally, the image database has a set of images, i.e., multimedia objects,

$$MMS = \{I_1, I_2, ..., I_m\}$$

where I_i is an image object and each image object is described by three feature vectors, $I_i = \{C_i, T_i, S_i\}$, where C_i is color histogram vector, T_i is texture information vector, S_i is shape information vector of image object i. Each feature vector for image I_i can be defined as follows:

$$C_i = < c_{i1}, ..., c_{in} >$$
$$T_i = < t_{i1}, ..., t_{ip} >$$
$$S_i = < s_{i1}, ..., s_{ir} >$$

where n, p and r are the number of entries in the feature vectors. So, an image can be represented as follows:

$$I_i = \{ < c_{i1}, ..., c_{in} >, < t_{i1}, ..., t_{ip} >, < s_{i1}, ..., s_{ir} > \}$$

Alternatively, we can show the image database with their corresponding features as a matrix as follows:

$$MMS = \begin{bmatrix} C_1 & T_1 & S_1 \\ C_2 & T_2 & V_2 \\ ... & ... & ... \\ C_m & T_m & S_m \end{bmatrix}$$

Partitioning can be done horizontally (row-wise) or vertically (column-wise). In this study, we focus on vertical partitioning due to the main issues of multimedia information retrieval (e.g., feature extraction and feature indexing). We propose vertical partitioning of data over the processors so that different feature algorithms and different indexing schemes can be applied to each feature.

2.2 Architecture

Figure 1 shows a typical architecture of PAMIR with one master and three slaves: one for color, one for texture and one for shape. Color vectors of all objects are stored in first slave, texture vectors of all images are in second slave and so on. A typical query can be processed as follows. The master processor gets the query via interacting by user and sends the request to slaves. Slaves process the query request concurrently and then master collects the each slave's response. Finally, it merges and outputs the results to user.

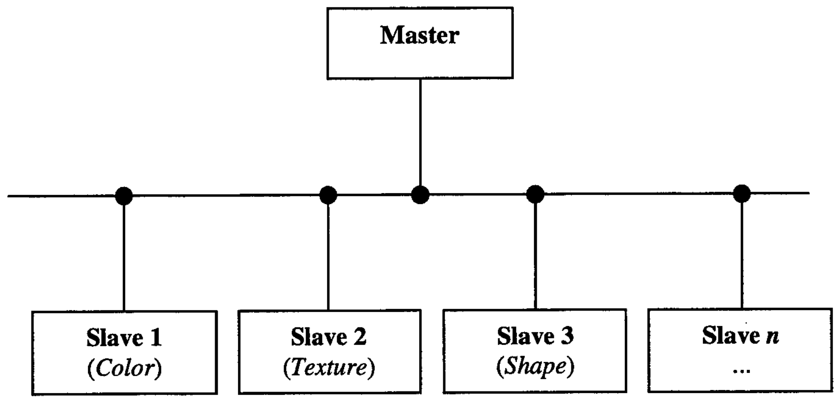


Figure 1. PAMIR architecture

When a new object is to be inserted into the database, firstly it is necessary to extract the feature vectors of object. To do this, object is broadcasted to all slaves. Then, each processor extracts its respective feature vector applying proper algorithms and stores the vector in its own disk. Each processor can use different indexing methodology.

If we process a complex query of three predicates sequentially in one machine, the expected query processing time will be the following;

$$T_{Serial} = (T_{S1} + T_{S2} + T_{S3}) + T_{Combine}$$

The time in a parallel environment like in PAMIR, the total time for query processing will become as follows;

$$T_{Parallel} = \max(T_{S1}, T_{S2}, T_{S3}) + T_{Combine}$$

where T_{S1}, T_{S2} and T_{S3} are the search times for respective sub-queries and $T_{Combine}$ is the time for combining the results from each sub-query. It is clear that $T_{Serial} > T_{Parallel}$ assuming that the communication time is a small fraction of the total query time for a large number of query. However, there is still an open problem for this kind of system for balancing the loads of the each processor for both feature extraction and query processing phase. The total time for both of the phases can be further improved by balancing the loads.

2.3 Query Processing

A Multimedia Information Retrieval Systems must be enable to find all objects in the region which is defined with a specific distance from a given query object in large linearly ordered domains of attribute values. For an image database, a typical query can be posed in natural language as, *"find all images having the red circle"*. In this example, the query has two predicates: one for color feature (red) and the other one for shape feature (circle). However, these feature vectors describing different media contents have different properties, i.e., different dimensionality and similarity or distance measure metrics.

If a multimedia query is formed with more than one predicate, as in above example, each of which corresponds a single feature then such queries are called *complex queries*. Complex queries may have a different similarity function from single feature queries. This is because the user wants to retrieve the best matches to the whole query. The similarity function of complex queries must be a combined function that compares all features available in the query. Fagin's A_0 algorithm provides a solution for multiple features at the same time. Based on Fagin's algorithm, some other researchers offered some improvements [NR99][GBK00]. We will explain these three algorithms and their parallel versions in the following sections.

Fagin's A_0 Algorithm

For a given complex query, (Color = "Red") $\wedge$ (Shape = "Round"), the solution set of this query is a "graded" (or "fuzzy") set. A graded set is a set that is given by some pairs such as (x, g), where x is an object, and g is a real number in the interval [0,1]. It can also be considered that a graded set is a sorted list, where the objects are sorted by their grades.

For each sub-query that handles a single feature predicate, a grade is assigned to each object. The grade represents the degree of fulfillment of object on this sub-query, where the larger the grade is, the better the match. In particular, a grade of 1 represents a perfect match. For traditional database queries, the grade for each object is either 0 or 1, where 0 means that the query is false about the object, and 1 means that the query is true about the object. For multimedia queries, grades may be intermediate values between 0 and 1. Boolean combinations of sub-queries are processed by some "aggregation functions" that assign a grade to a fuzzy conjunction, as a function of the grades assigned to the conjuncts.

If x is an object and Q is a query, let $\mu_Q(x)$ the grade of x against the query Q. Based on definitions and boolean combinations of sub-queries are defined by the standard fuzzy logic given below;

Conjunction rule : $\mu_{A\wedge B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
Disjunction rule: $\mu_{A\vee B}(x) = \max\{\mu_A(x), \mu_B(x)\}$
Negation rule: $\mu_{\neg A}(x) = 1 - \mu_A(x)$

These are the 2-ary aggregation functions because there are 2 queries (A and B) combined. What else if more than 2 queries? Such these queries are called m-ary queries. Let $F(A_1, \ldots, A_m)$ be an m-ary query, where $A_1, \ldots, A_m$ are query terms. The grade of each object under m-ary query is defined as $\mu_{F(A1, \ldots, Am)}$. The standard fuzzy logic semantics of the conjunction $A_1\wedge \ldots \wedge A_m$ is given by defining

$$\mu_{A1\wedge\ldots\wedge Am}(x) = \min\{\mu_{A1}(x), \ldots, \mu_{Am}(x)\},$$

for each object x. Let t be an m-ary aggregation function.The m-ary query

$F_t(A_1, \ldots, A_m)$ is defined by

$$\mu_{Ft(A1,\ldots,Am)}(x) = \mathrm{t}\,(\mu_{A1}(x), \ldots, \mu_{Am}(x))\ .$$

Fagin's A_0 algorithm was used in Garlic project, for combining multiple features. Both conjunctive and disjunctive queries can be processed using A_0 algorithm. The algorithm returns the top k answers for a query with multiple features $F(A_1, \ldots, A_m)$, which is denoted by Q. It consists of three phases: sorted access, random access and computation phase.

1. There is a subsystem i that evaluates the sub-query A_i. For each i the corresponding subsystem outputs a set one by one in sorted order based on grade, the graded set consisting of all pairs $(x, \mu_{Ai}(x))$, where as before x is an object and $\mu_{Ai}(x)$ is the grade of x under query A_i.

   This process is repeated until the intersection of the m lists is of size at least k. That is, repeat until there is a set L of at least k objects such that each subsystem has output all of the members of L.

2. For each object x that has been seen, do a random access to each subsystem j to find $\mu_{Aj}(x)$. If x is in the graded set of sub-query A_j, there is no need to do a random access for x in subsystem j.

3. Compute the grade $\mu_Q(x) = t(\mu_{A1}(x), \ldots, \mu_{Am}(x))$ for each object x that has been seen. Sort the objects on their grades and form a result set. The output is then selected from this set with the top k objects.

Multi-Step Query Processing Algorithm

This algorithm was proposed in [NR99] and claimed that it performs better than Fagin's algorithm, because it requires fewer database accesses. It was shown that, to improve Fagin's algorithm, combining function must satisfy two additional properties defined below.

Definiton 1 *Lower bounding property*: A combining function t for the objects x and x' under the query $A_1 \wedge A_2 \wedge \ldots \wedge A_m$ satisfies the lower bounding property iff

$$\exists i \forall j : \mu_{Ai}(x) \leq \mu_{Aj}(x') \Rightarrow t(\mu_{A1}(x), \ldots, \mu_{Am}(x)) \ \leq\ t(\mu_{A1}(x'), \ldots, \mu_{Am}(x'))$$

Definiton 2 *Upper bounding property*: A combining function s for the objects x and x' under the query $A_1 \vee A_2 \vee \ldots \vee A_m$ satisfies the upper bounding property iff

$$\exists i \forall j : \mu_{Ai}(x) \geq \mu_{Aj}(x') \Rightarrow s(\mu_{A1}(x), \ldots, \mu_{Am}(x)) \ \geq\ s(\mu_{A1}(x'), \ldots, \mu_{Am}(x'))$$

Again, the standard rules of fuzzy logic for conjunction, disjunction and negation are used from combining functions. The standard fuzzy rule "min" satisfies the lower bounding property while the standard fuzzy rule "max" satisfies the upper bounding property.

Multi-step query processing algorithm returns the top k objects for a query $Q=F_t(A_1,\ldots, A_m)$. Each subsystem i, where $1 \leq i \leq m$, returns the matching objects in sorted order. The termination condition is to fill the result list with at least k objects. This process is an iterative process.

1. For each query term A_i, request the subsystem i to return the next object x. Thus, each subsystem i outputs the graded set of pairs $(x, \mu_{Ai}(x))$, where x is an object in the database and $\mu_{Ai}(x)$ is the similarity value of x under A_i.

2. For each object x returned by the subsystem i in the current iteration, do random access to subsystems j, $i \neq j$, to find $\mu_{Aj}(x)$.

3. Compute the threshold grade, t_h for this iteration, $t_h = t(\mu_{Ai}(x_i), \ldots, \mu_{Am}(x_m))$ where x_i is the element retrieved by subsystem i in the current iteration.

4. Compute the grade $\mu_Q(x) = t(\mu_{Ai}(x), \ldots, \mu_{Am}(x))$ for each object x retrieved in this iteration. Update Y, the set containing all objects that have grade $\mu_Q(x) \geq t_h$.

5. Go to next iteration (repeat steps 2 to 5) until the set Y has k objects.

6. Output the graded set $\{(x, \mu_Q(x)) | x \in Y\}$

Their theorem on this algorithm is "*For every query $F_t(A_1, \ldots, A_m)$, the multi-step algorithm correctly returns the top k answers, if t satisfies the bounding properties*". From their observations, Fagin's algorithm requires more database accesses. Because, when the lower and upper bound properties are satisfied by the combining function used, the number of random accesses for non-available grade values can be reduced.

Quick-Combine Algorithm

Quick-Combine algorithm proposed in [GBK00] is similar to multi-step algorithm in the property of iterative process. But in this case, instead of computing a threshold value, present-top scored object is found in each step. Termination condition is to retrieve a list that has at least k objects. The number of necessary database accesses can be minimized with this new test of termination. Not only the information of ranks in output streams, but also the scores which are assigned to all objects in each output stream and the specific form of the combining function is used for this new test.

The following algorithm returns the top answer (k = 1) to any combined query consisting of n atomic sub-queries $q_1,\ldots,q_n$ aggregated using a monotone combining function F. Let x be an object and $s_i(x)$ be the score of x under sub-query q_i. An object occurring in the result set of sub-query q_i on rank j will be denoted $r_i(j)$.

1. For each sub-query compute an atomic output stream consisting of pairs $(x, s_i(x))$ in descending order based on score and get some first elements.

2. For each object output by a stream that has not already been seen, get the missing scores for every sub-query by random access and compute its aggregated score $S(x) = F(s_1(x), \dots, s_n(x))$.

3. Check if the present top-scored object o_{top} is the best object of the database:

   Compare the aggregated score $S(o_{top})$ to the value of F for the minimum scores for each sub-query that have been returned so far. Test:

   $$S(o_{top}) \geq F(s_1(r_1(z_1)), \dots, s_n(r_n(z_n))) \qquad (1)$$

   where z_i is the lowest rank that has already been seen in the output stream of q_i.

4. If inequality 1 holds, o_{top} can be returned as top object of the whole database. If inequality y 1 does not hold, more elements of the output streams have to be evaluated. Therefore get the next elements of the streams and proceed as in step 2 with the newly seen objects.

2.4 Combining Complex Queries in PAMIR

In PAMIR, complex queries are processed based on the algorithms mentioned before. All three algorithms have been adapted into PAMIR environment to retrieve the result of complex queries. The steps that we followed while processing queries are given in Algorithm 1, 2 and 3.

Algorithm 1. Fagin's A_0 Algorithm

Step 1. Master gets the user query in the form of boolean combination of sub-queries, i.e. Q = (Color="Red" and Texture = "Fine" and Shape = "Circular")

Step 2. Master divides the whole query into sub queries according to the predicates of each feature, i.e. Q_1 = (Color = "Red"), Q_2=(Texture="Fine"), Q_3=(Shape="Circular")

Step 3. Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has k elements for color, texture and shape are required. This step meets the sorted access phase of Fagin's algorithm.

Step 4. Each slave evaluates its own sub-query and sends the result sets back to master.

Step 5. Master merges all lists and forms a single set. The objects in this set may not have similarity values for all features. That is, there may be non-available grade values for object. Master determines the non-available grade values of each object and send a request for grade value to the corresponding slave processor, i.e. non-available grade values for color feature are sent to slave 1, non-available grade values for texture feature are sent to slave 2, non-available grade values for shape feature are sent to slave 3. This step meets the random access phase of Fagin's algorithm.

Step 6. Each slave gets its request and begins to compute the similarity values for objects. Then, they send the results back to master.

Step 7. Master computes the similarity values of each object under the overall query (Q) and sorts them on their similarities. Finally, it outputs the top k objects (highest k objects) to user. Last step meets the computation phase of Fagin's algorithm.

Algorithm 2. Multi-Step Query Processing Algorithm

```
Step 1.  Master gets the user query in the form of boolean combination of sub queries, i.e. $Q$ = (Color = "Red" and Texture = "Fine" and Shape = "Circular")
Step 2.  Master divides the whole query into sub queries according to the predicates of each feature, i.e. $Q_1$ = (Color = "Red"), $Q_2$ = (Texture = "Fine"), $Q_3$ = (Shape = "Circular")
Step 3.  Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has $k$ elements for color, texture and shape are required.
Step 4.  Each slave evaluates its own sub-query and sends the result sets back to master.
Step 5.  Master doesn't merge all lists. It picks just one object from returned list. And sends this object to all slaves except the subsystem that the object belongs it to compute the missing grade values. This phase includes random access for one object.
Step 6.  Master computes a threshold value based on all subsystems. Then for selected object the grade value of overall query is computed from the grade value under all subsystems. If the grade of this object is bigger than threshold value then add it to the final list.
Step 7.  Go to step 5 and stop until the final list has at least $k$ objects.
Step 8.  Finally, show the graded set of $k$ objects to the user.
```

Algorithm 3. Quick-Combine Algorithm

```
Step 1.  Master gets the user query in the form of boolean combination of sub queries, i.e. $Q$ = (Color = "Red" and Texture = "Fine" and Shape = "Circular")
Step 2.  Master divides the whole query into sub queries according to the predicates of each feature, i.e. $Q_1$ = (Color = "Red"), $Q_2$ = (Texture = "Fine"), $Q_3$ = (Shape = "Circular")
Step 3.  Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has $k$ elements for color, texture and shape are required.
Step 4.  Each slave evaluates its own sub-query and sends the result sets back to master.
Step 5.  For returned objects, master picks $p$ objects among them where p is a suitable natural number. It sends these objects to all slaves except the subsystem that the object belongs it to compute the missing score for every sub query by random access. For each new object there are $(n - 1)$ random accesses necessary.
Step 6.  Master checks if the $k$ top-scored objects are the best $k$ objects of the database. It compares the aggregated score of the present k top-scored objects to the value of the combining function with the lowest scores seen from each feature. Then, it checks if there are at least $k$ objects whose aggregated score is larger or equal than the aggregated minimum scores per feature.
Step 7.  Master checks if the $k$ top-scores of $k$ objects are bigger than the minimum scores then the $k$ top-scored objects can be returned as top objects of the whole database. I this case, add them into the final list and go to step 8. If not, more elements of the atomic output streams have to be evaluated. In this case, go to step 5 for next $p$ objects and stop until the final list has at least $k$ objects.
Step 8.  Finally, show the graded set of $k$ objects to the user.
```

3 Performance Evaluation

The current implementation of PAMIR is built on top of four Pentium MMX based workstations (one master and three slaves), running at 233 Mhz processor speed. Master has 128 MB memory, and the rest have 64 MB. The workstations are linked via a 100 Mbit/s Fast Ethernet under the operating system of Linux. The communication among the processors provided using MPI (message passing interface).

The system is tested on a collection of 68040 images from various categories. This dataset contains image from a Corel image collection. Images are described with three features based on the color histogram, color moments, and co-occurrence texture.

For color histogram, HSV color space is divided into 32 subspaces (32 colors: 8 ranges of H and 4 ranges of S). In the resulting histogram, the value in each dimension in a color histogram of an image is the density of each color in the entire image. Histogram intersection was used to measure the similarity between two images.

Color moments, we obtained 9 dimensional vectors (3 x 3) (one for each of H,S, and V in HSV color space) mean, standard deviation, and skewness. Euclidean distance between color moments of two images is used to represent the dis-similarity (distance) between two images.

The last feature used in our experimentation is for texture. Texture features are extracted by co-occurrence matrix, resulting 16 dimensions (4 x 4). Images are first converted to 16 gray-scale images and co-occurrence matrix in 4 directions is computed (horizontal, vertical, and two diagonal directions). The 16 values are (one for each direction) second Angular Moment, Contrast, Inverse Difference Moment, and Entropy. Euclidean distance between co-occurrence matrices of two images is used to measure the dis-similarity (distance) between two images.

In order to evaluate the system, we have done some experimentation to test different combining algorithms for complex queries on our parallel environment for varying *k* values for a 100 successive queries. Figure-2 shows the results we obtained from our experimentation. As *k* value increases Multi-step and Quick-combine algorithms are seemed to be approaching to Fagin's algorithm due to increased communication cost among processors.

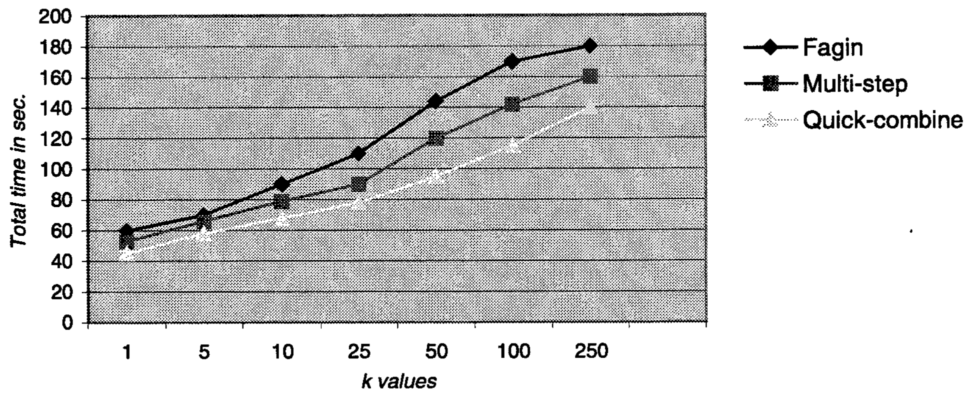


Figure 2. Result of combining algorithms comparison experiment.

While processing queries, we have added one more features in each step and observed the total time for 100 successive queries. As we added new features to PAMIR the total time has been increased slightly due to increased communication cost. So, it can be said that the PAMIR is almost scalable with increasing number of features. We are planning to observe how scalability is changed as more features are added.

4 Conclusion

This paper presents the design, implementation, and evaluation of a parallel multimedia information retrieval system called PAMIR to exploit the I/O and computation parallelisms specifically for multi-feature complex query. We have successfully implemented the first PAMIR prototype on a network of Pentium workstations, and carried out a comprehensive performance evaluation of the prototype against a database consisting of 68040 images where each image has been described by three features, color histogram, color moments and co-occurence texture.

In complex query processing phase, we have evaluated Fagin's, multi-step and quick-combine in our parallel environment. The performance evaluation experiments showed that quick-combine algorithm outperforms the others. Stream-combine algorithm is also a good solution if the order in final result set is not important. We have presented both all experimentation results and parallel version of respective algorithms for PAMIR environment.

PAMIR is a prototype system for parallel Multimedia Information Retrieval, which is designed to address issues for both in *feature extraction* and *feature indexing* phases and it helps to process complex queries. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both search and insertion process of multimedia retrieval. PAMIR provides a good speed up in both database population and query processing response time, and it is scalable with the increasing number of features used in the system. In longer term, we expect the PAMIR project to lead us into new research in many dimensions, including query processing technologies, user interfaces, parallel similarity indexing and more.

References

[CH+95] W.F.Cody,L.M.Haas,W.Niblack, M.Arya,M.J.Carey, R. Fagin, M. Flickner, D. Lee, D. Petkovic, P. M. Schwarz,J. Thomas, M. T. Roth, J. H. Williams, and E. L. Wim-mers. "Querying multimedia data from multiple repositories by content: the Garlic project", *Proceedings of Third Working Conference on Visual Database Systems, VDB-3*, Lausanne, Switzerland, March 1995.

[CP00] Paolo Ciaccia, Marco Patella, "The M2-tree: Processing Complex Multi-feature Queries with just One Index", First Delos Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, Swithzerland, 2000.

[CPZ97] Paolo Ciaccia, Marco Patella, Pavel Zezula, "M-tree: An efficient access method for similarity search in metric spaces", *Proceedings of the 23rd VLDB International Conference*, Athens, Greece, September 1997.

[CPZ99] Paolo Ciaccia, Marco Patella, Pavel Zezula, "Processing Complex Similarity Queries with Distance-based Access Methods", *Proceedings of the 6th EDBT International Conference*, Valencia, Spain, March, 1998.

[Fag99] Ronald Fagin, "Combining fuzzy information from multiple systems", *Journal of Computer and System Sciences*, Vol. 58, pp. 83-99, 1999.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling, "Optimizing Multi-Feature Queries for Image Databases", In *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, pp. 419-428, Cairo, Egypt, 2000.

[GBK01] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling, "Toward Efficient Multi-feature Queries in Heterogeneous Environments", Proc. Of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2001), Las Vegas, USA, 2001.

[NR99] Surya Nepal and M.V. Ramakrishna, "Query Processing Issues in Image(multimedia) Databases", *Proceedings of the 15th International Conference on Data Engineering* , Valencia, Spain, March, 1999.

[WH+99] Edward L. Wimmers, Laura M. Haas, Mary Tork Roth, Cristoph Braendli, "Using Fagin's Algorithm for Merging Ranked Results in Multimedia Middleware", *Proceddings of 4th IFCIS International Conference on Cooperative Information Systems*, Edinburgh, SCOTLAND, September 1999.

Implementing Relevance Feedback Techniques for Large Image Collections Efficiently

Klemens Böhm **Ana Stojanovic** **Roger Weber**

Institute of Information Systems, ETH Zentrum, 8092 Zurich, Switzerland
`{boehm,stojanov,weber}@inf.ethz.ch`

Abstract

Relevance Feedback is a powerful technique to raise the quality of search results in image databases. A number of relevance feedback models have been proposed. Frequently, these solutions apparently do not scale with the number of images. Another issue is that some relevance feedback models support only simple similarity queries, but not similarity queries with a more complex structure, e.g., more than one reference image. This article in turn describes the design and implementation of a *relevance feedback engine* that supports relevance feedback in its full generality. To this end, we have developed a query language for similarity search with the following characteristics: it is expressive enough to map user feedback to a statement in the language for any of the relevance feedback models under consideration here. At the same time, it is not excessively complex and allows for an extremely efficient implementation, even for large data sets.

1 Introduction

Similarity search for images plays a key role in a large number of multimedia applications. Examples can be found in astronomy, entertainment, education, journalism, advertisement, and cultural heritage. Typical approaches for image retrieval are either keyword-based or content-based. In this context, *keyword-based* retrieval identifies relevant images by applying textual retrieval methods to the annotations or meta data of the images. On the other hand, *content-based* retrieval looks for similar images based on statistical information on the raw image data. This includes color, texture and shape features [13, 6] as well as high level classification features (e.g. to detect faces in an image). In the past, most work on image similarity search has focused either on the extraction of characteristic features from the images and the definition of effective similarity measurements on top of them, or the development of efficient index structures for simple similarity search, i.e. efficient nearest neighbor search methods. However, only little work has addressed the *efficient* evaluation of *effective* similarity measurements so far [3, 5, 2], i.e. fast evaluation of complex queries consisting of several reference images and several features (including textual features). Moreover, most of the existing approaches do not scale well with the database size. For instance, our implementation of Fagin's A0-Algorithm often resulted in query times well above 100 seconds (see also [15]).
As a consequence, image retrieval systems either perform simple similarity searches and are able to deal efficiently with huge image collections, or they apply complex and effective similarity measurements but are restricted to small database sizes. Moreover, most image retrieval systems do not allow for *relevance feedback*. It is a concept that has been introduced for text retrieval to improve the quality of

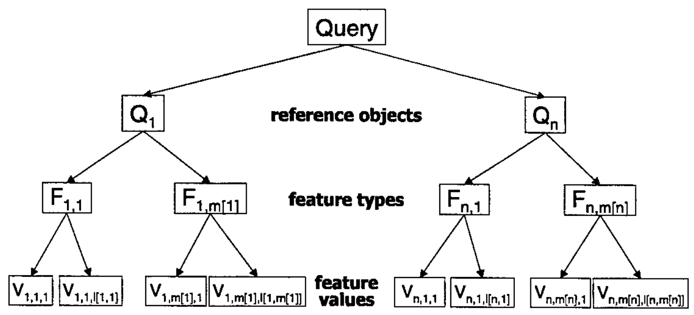


Figure 1: Query Model

search results. A search with relevance feedback consists of several steps. The user judges the search results after each step. The system takes this feedback into account in subsequent steps to come up with results of higher quality. The problems with relevance feedback for image retrieval are the incapability of the retrieval system to handle complex queries (which often result from the feedback step) or the long duration of query evaluation, which is in the way of interactive search.

The objective of this article is to describe our image similarity search system. It carries out complex similarity search efficiently and supports relevance feedback in its full generality. Our query model is a reasonable compromise between effectiveness and efficiency: we restrict our model to two-level queries consisting of a number of reference images. Each such image can define its own similarity measurement based on a set of feature types (see Figure 1). We have implemented an efficient search technique for this query model called GeVAS (generalized VA-File based search) [1]. In combination with parallelization [14], this method identifies the most similar images in a database of more than 230,000 images and with complex similarity queries within a few seconds (less than 5s). Our image retrieval system supports relevance feedback by generating statements in our query model, and the mapping of feedback to query statements reflects the respective feedback model. We show that this is feasible with any of the models considered [8, 9].

The remainder of this article has the following structure: Section 2 summarizes our query model and the implementation of the GeVAS search technique. Section 3 provides a comprehensive overview of the relevance feedback models used in the context of image similarity search. Section 4 reviews related work. Section 5 concludes.

2 Query Model for Similarity Search

Our apprach to relevance feedback maps user feedback to a statement in our query model for similarity search. This model must be appropriate for our purpose, i.e., (1) it must be sufficiently expressive to reflect user feedback with any of the feedback models under consideration, and (2) it should not be unnecessarily complex and allow for an efficient implementation. In the remainder of this section, we summarize our previous work [1], namely the query model and its semantics as well as the efficient evaluation of complex queries with the *Generalized VA-File Based Search Technique* (GeVAS).

2.1 Query Structure

A common approach to measure similarity between an image in the collection and a reference image is to extract so-called *features*, both from the images in the collection and from the reference images. A feature describes a particular characteristic of an image, e.g., its color distribution. Typically, a feature is a vector in a high-dimensional space with a dimensionality varying between 5 and 500. This space is typically referred to as *feature space*. Two images are assumed to be similar with regard to a certain feature type if the distance between their feature vectors is small.
Figure 1 displays the general structure of a query. A query consists of *n reference objects*. For each reference object Q_i, there may be an arbitrary number of feature types $(F_{i,1}, \ldots, F_{i,m[i]})$. Finally, the point $(v_{i,j,1}, \ldots, v_{i,j,l[i,j]})$ is the representation of query object Q_i in feature space $F_{i,j}$. For the sake of presentation, Figure 1 omits the weights at the various levels of the query model. In the following, an *atomic sub-query* of multi-feature multi-object query is a sub-query with one reference image over one feature type. The following examples provide some intuition regarding the structure of a query.

Example 1 (Complex Query) *A user is looking for an image of a black cow, but he only has a picture of a brown cow and one of a black horse at hand. The corresponding query has two reference objects Q_1 (cow) and Q_2 (horse). There is one feature type for each image: $F_{1,1}$ is the shape feature of Q_1, $F_{2,1}$ the color feature of Q_2.*

Example 2 (Relevance Feedback) *Usually, a user would not type in a complex query as insinuated in Example 1. Rather, he tells the system which images he likes and for what reasons. To continue the above example, his initial query might have returned an image A with a black horse and image B with a brown cow. The user would rate the images as follows: image A contains an object with the* right color *but* wrong shape, *and image B has the* right shape *but the* wrong color. *The resulting query is equal to the one in Example 1 but is now generated by the system. Note that relevance feedback based on Rocchio [8] will often not suffice. In the current example, Rocchio would produce artificial vectors for the shape and color feature that lie between the two vectors for the images.*

2.2 Query Semantics

Distances. To state what a similarity query stands for, we introduce the notions of *feature distance*, *reference-object distance*, and *query distance*. The feature distance measures the dissimilarity of a reference object and a data object regarding a single feature type. The i-th reference-object distance of a data object is a *combined distance* to the i-th reference object with regard to the feature representations $F_{i,1}, \ldots, F_{i,m[i]}$. Moreover, we allow to weight the individual features to reflect their importance for the current information need. Finally, the query distance is a combined distance over all reference objects. We also allow for weighting on this level of the query model.
Distance Functions. Each object of the database and each reference object is mapped to a set of feature representations. The l-th feature representation of an object lies in a d_l-dimensional feature space. A distance measure $\delta_l(\mathbf{p}, \mathbf{q})$ defines the distance between two points in this space. A common measure is the (weighted) Minkowski metric L_s:

$$L_s: \; \delta_l(\mathbf{p}, \mathbf{q}) \; = \; \sqrt[s]{\sum_{j=1}^{d_l} w_j \cdot |p_j - q_j|^s} \tag{2.1}$$

Distance-Combining Functions. To describe the reference-object distance and the query distance, we introduce the notion of *distance-combining function* (Δ). It combines a set of distances to an overall distance and must be monotonic in all arguments, i.e. $\Delta(x_1, \ldots, x_m) \leq \Delta(x'_1, \ldots, x'_m)$ if $x_i \leq x'_i$ for all i. Note that monotonicity is a natural requirement: if all distances become larger, the similarity decreases. The following distance-combining functions implement different ways of combining distances. The example is such that they combine reference-object distances to a query distance.

Example 3 (Fuzzy-And, Fuzzy-Or, Weighted Average) *Given a set of n reference objects,* Fuzzy-And *implements the idea that a data object should be close to all of them. This leads to the maximum function* (max) *for* Δ, *i.e.:* $\delta(P) = \Delta_{and}(\delta_1(P), \ldots, \delta_n(P)) = \max_{i=1}^{n} \delta_i(P)$.
If the reference objects represent a set of possibilities, and each of them is acceptable, the Fuzzy-Or *distance-combining function would be more appropriate. Rather than* $\Delta = \max$, *it uses* $\Delta = \min$.
Finally, let $\mathbf{w} \in \mathbb{R}^n$ *be the weights of the individual distances such that* $\sum_{i=1}^{n} w_i = 1$. *Then* Weighted Average *is given as* $\Delta = \sum_{i=1}^{n} w_i \cdot \delta_i$.

Normalization. Combining feature distances as described so far is not sufficient since the meaning of a distance value depends on the distance distribution. For example, assume we would deploy Fuzzy-Or on a distance from the feature space $[0,1]^{10}$ and on one from the feature space $[-10,10]^{100}$. As distances in the first space are much smaller than in the later one, Fuzzy-Or yields the distance from the first space in almost all cases. This would mean that the second feature did not matter with regard to overall similarity. We normalize distances with a well-chosen normalization function for each distance measure to avoid such situations.
Weighting. Some relevance-feedback models further require weighting of individual reference objects, feature types and dimensions within a single feature vector. The distance function implements dimension weights, e.g., w_i in Equation 2.1. Weighting reference objects and feature types is straightforward with Weighted Average. But this is not the case with Fuzzy-And or Fuzzy-Or. Fagin et al. solved this problem for a slightly different setting, and we deploy their solution in our context (cf. [4]).
Distances and Scores. While distances measure dissimilarity, it would also be possible to work with *scores* that measure the similarity. Scores are *normalized* by definition, i.e., from the interval $[0,1]$ (1=identity, 0=no similarity). We deploy a *correspondence function* [2] to map query distances to overall scores. Hence, unlike [3, 2, 15], we do not need to deal with scores at the intermediate levels.

2.3 Query Evaluation

Our earlier work has combined/generalized existing approaches, resulting in a set of techniques called Generalized VA-File-based Search (GeVAS) [1]. An extensive evaluation of our techniques has shown that GeVAS gives rise to a significant improvement over other approaches (usually by more than a factor of 10). Figure 3 shows elapsed times for multi-object queries and multi-feature queries within our database containing around 230,000 images ($k = 15$ means that we search for the 15 most similar objects to the query). On the left hand side, the response times for multi-object, single-feature (a 45-dimensional color feature) queries of GeVAS are compared with some competitive methods ([2] applied to M-Tree, R*-Tree, and SR-Tree as well as [3] based on VA-File). With around 10 reference objects, GeVAS outperforms all other approaches by more than a factor of 10. This kind of query is important with the query extension method discussed later in Section 3.6. Figure 3 (b) plots the response times for single-object, multi-feature queries. In the figure, F_x denotes a feature type with x dimensions.

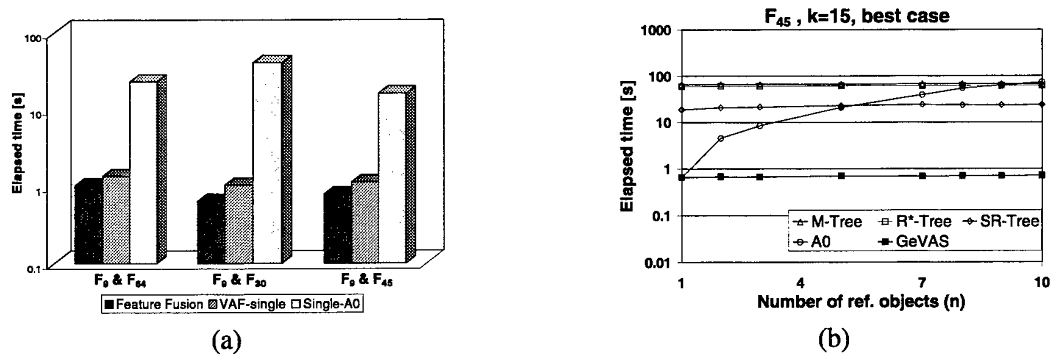


Figure 2: Elapsed times for (a) multi-object queries, and (b) multi-feature queries.

The left-most bars (Feature Fusion) represent optimal search costs with the VA-File by combining the feature vectors a priori. The bars in the middle represent search costs with GeVAS; in this case, the features are combined at evaluation time. The right-most bars correspond to the search costs of the A0-Algorithm using VA-Files as base search structures [3]. Again, the figure shows the superiority of GeVAS: the A0-algorithm is always more than a factor of 10 slower than GeVAS. Moreover, GeVAS almost reaches the optimal case with pre-composed feature vectors. In practice, however, this optimal scheme is not feasible since the number of combinations with around 20 feature types is by far too large (2^{20}).

In the general case (see [1]), query response times grow linearly with the database size and the complexity of the query, i.e. the number of features and the number of reference objects. In most relevant cases however, depending on the distance functions and the chosen distance-combining function, our technique has a constant complexity in the number of reference objects.

3 Relevance Feedback for Image Similarity Search - State-of-the-Art and New Directions

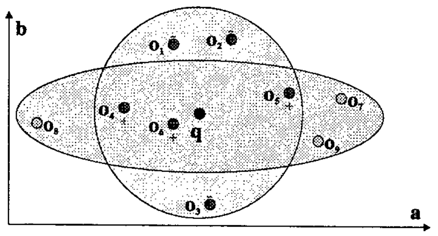
This section deals with various aspects of relevance feedback. We first classify user feedback in various dimensions. We then give a summary of existing relevance feedback models from literature, i.e., possible interpretations of user feedback.

3.1 Dimensions of User Feedback

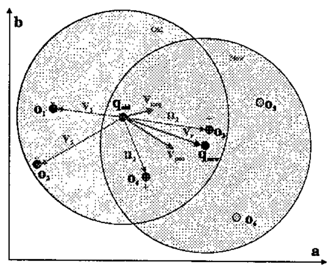
We can distinguish user feedback in the following dimensions:

Granularity of feedback. User feedback may either refer to the image as a whole, it may refer to a particular feature, or it may refer to an individual aspect of the feature. An example for the last case for feature type 'color' is feedback on the individual *RGB values* of the image.

Range of feedback predicates. Another dimension is the number of possible feedback values. Possible values are 'relevant'\'not relevant' on the one hand and a continuous scale from -1 to 1 on the other hand, among others. Another differentiation is to facilitate only explicit positive



(a)



(b)

Figure 3: (a) Example of dimension weighting (Euclidean metric). (b) Illustration of Rocchio.

feedback, i.e., have feedback values 'relevant' and 'indifferent' on the one hand, as opposed to 'relevant', 'not relevant' and 'indifferent' on the other hand.

Obviously, the tradeoff is between user effort and preciseness of feedback. Preliminary experiments indicate that users favor the coarser variants in both dimensions. We use a homogeneous internal representation of user feedback with regard to implementation. I.e., we map coarse feedback internally to a finer, equivalent representation.

We have conducted an extensive survey of literature on relevance feedback models used for image retrieval. The main alternatives encountered are dimension weighting, feature weighting, query weighting, query point movement, and query extension. The following subsections discuss these models.

3.2 Dimension weighting

Recall that each image is represented by a number of feature vectors in different feature spaces. All dimensions have the same importance (weight), with the common distance metrics. The principle of dimension weighting [9] is to modify the weights of the dimensions according to the user feedback.

Example 4 *Figure 3.1 serves as an illustration. For the sake of simplicity, assume that the query has one subquery with one feature type.* $\mathbf{q}$ *is the original query point, and* $\mathbf{o}_1,\ldots,\mathbf{o}_6$ *are the items returned in the first step of the search.* $\mathbf{o}_1,\ldots,\mathbf{o}_3$ *are the items judged relevant by the user, and* $\mathbf{o}_4,\ldots,\mathbf{o}_6$ *are not relevant. Obviously, Dimension b discriminates better between relevant and non-relevant items than Dimension a. With dimension weighting, the relevance feedback component adjusts the dimension weights. In this case, it would generate a new query which corresponds to the ellipse.*

More formally, let i be the number of a dimension, and let $\sigma(i)$ denote the standard deviation of the i-th components of the *relevant* data points. Obviously, the smaller the standard deviation for a dimension, the better the dimension discriminates between relevant and non-relevant items. Consequently, if the standard deviation is small, the weight on the dimension should be large, i.e. $\bar{w}(i) = 1/\sigma(i)$. These weights further need to be normalized to avoid numerical instabilities.

Dimension weighting requires at least two relevant objects. Otherwise the standard deviation is not meaningful. Furthermore, a larger number of objects judged by the user facilitates a more accurate weighting, because it reduces the influence of a single object.

Finally, dimension weighting for complex queries in our system looks at each atomic subquery in isolation: dimension weighting is carried out in the respective feature space and the reference object as described above for simple queries.

3.3 Feature Weighting

Dimension weighting adjusts the weights of individual dimensions. Feature weighting in turn adjusts the weights of features as a whole [9]. The input of feature weighting is the result list of the overall query and the user feedback, but also the result lists of each atomic sub-query. The more relevant images from the result list of an atomic query appear in the final result, the higher should be the weight of the respective atomic query. More formally, let A_i be the objects in the result list of the i-th atomic sub-query, F be the objects in the overall results, R be the objects judged relevant, and N be the objects judged non-relevant. The feature weight $\bar{w}(i)$ of the i-th atomic sub-query is then as follows:

$$\bar{w}(i) = max(k_1 \cdot |A_i \cap R \cap F| - k_2 \cdot |A_i \cap N \cap F|, 0) \tag{3.1}$$

where k_1, k_2 are parameters that specify the influence of the relevant and the non-relevant objects on the feature weight.
A few problems have arisen with our tests of the algorithm. A short result list rarely produces any matchings. Applying the algorithm in such a situation does not make sense. This feedback model also is much more expensive than dimension weighting from an implementation perspective. Since the model needs the result list of each atomic sub-query, it has to submit a number of single-object single-feature queries to generate these lists.

3.4 Query Weighting

In analogy to feature weighting, query weighting defines weights on sub-queries on the reference image level, i.e. it determines how well a reference image describes the information need of a user. Obviously, query weighting compares the result lists of the sub-queries at the reference image level with the final result, instead of the result lists of the atomic queries (at the feature type level).

3.5 Query Point Movement

Query Point Movement is a method whose implementation is feasible with a less powerful query model. We first describe the basic version of the algorithm, first published in [8], and then say how to extend it to finer feedback and to complex queries.
Recall that the vector space model represents the query and the database objects as points in a high-dimensional feature space. The model uses the positive feedback to move the query point towards the relevant points, and the negative feedback to move the query away from the non-relevant pointsThe following formula captures this.

$$\mathbf{q_{new}} = \mathbf{q_{old}} + \frac{\beta}{|R|} \sum_{\mathbf{r} \in R} \mathbf{r} - \frac{\gamma}{|N|} \sum_{\mathbf{n} \in N} \mathbf{n} \tag{3.2}$$

R and N are sets containing the relevant and the non-relevant objects, respectively. Finally, β and γ are constant factors reflecting the influence of relevant and non-relevant objects on the new query point. Normally $\beta \geq \gamma$: otherwise, the query may go too far away from the relevant objects in some cases. Good values for β and γ can be determined experimentally. We for our part have taken the values from [10], namely $\beta = 0.75$ and $\gamma = 0.25$. These values have been used successfully in text retrieval.

Example 5 *Consider the situation depicted in Figure 3.1. Query* $\mathbf{q_{old}}$ *has four nearest neighbors*

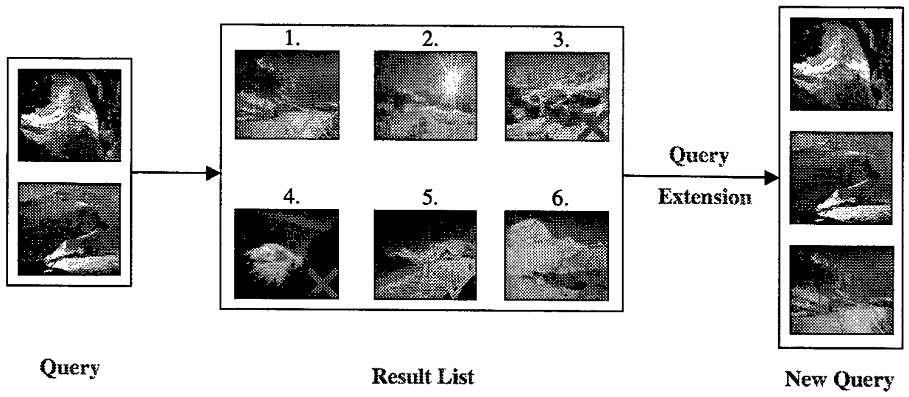


Figure 4: Example of a simple query extension method.

$\mathbf{o}_1 \ldots \mathbf{o}_4$. $\mathbf{o}_1$ *and* $\mathbf{o}_2$ *are non-relevant, but* $\mathbf{o}_3$ *and* $\mathbf{o}_4$ *are relevant.* $\mathbf{v}_{\mathbf{neg}}$ *is the shift of the query point resulting from the non-relevant objects,* $\mathbf{v}_{\mathbf{pos}}$ *the shift resulting from the relevant ones. The new query resulting from query point movement has other nearest neighbors, including objects* $\mathbf{o}_5$ *and* $\mathbf{o}_6$.

Objects are either relevant or non-relevant with the Rocchio model. But feedback can be more fine-grained (cf. Subsection 3.1). With our extension of the model, an object with a high feedback value, be it relevant, be it non-relevant, has a stronger influence on the new query point than the objects with smaller values. With multi-object multi-feature queries, we apply Rocchio's formula to each sub-query with a single reference image and a single feature type. Finally, note that the new query points no longer represent an existing reference image. Rather, each of them describes an artificial object that subsumes the characteristics of the reference image, the relevant images and the non-relevant images.

3.6 Query Extension

Query Extension takes the relevant images and simply adds them to the query as new sub-queries (using the same feature types as the reference images). Figure 4 serves as an illustration. In this example, only the highly relevant images, denoted by two hooks, extend the query.[1]

3.7 Improvements of Existing Relevance Feedback Techniques

Additional measures improve the feedback process and give rise to better results.

Blacklists. If the user has judged an image as non-relevant, it should not appear again in the result list of a subsequent step. We propose to set up a *blacklist*: it collects the IDs of the non-relevant images. If an image in a result list is an entry in the blacklist, it is removed from the result list.

Taking the user feedback from all steps into account. A search process may consist of several steps, and each step collects feedback information. Thus, the system typically does not only have the current user feedback, but also the one from the previous steps. Some methods (but not all of them) could use all the feedback information available.

[1]Image 1 has two hooks, Image 2 and 5 just one, i.e., these are the images judged relevant. Images 3, 4 and 6 have been crossed off as non-relevant.

The bottom line of this section and the previous one is that we have a relatively simple but fairly expressive query model for similarity search that is sufficient for the relevance feedback models addressed here. Moreover, we have an implementation of the query model that is based on the VA-File, and that is very efficient.

4 Related Work

There is a number of image retrieval systems that provide relevance feedback functionality. MARS is the most influential one with regard to our current work. It implements weighting of dimensions and features. However, the main differences are that our query model is more general (more than one reference object), and we had to extend the relevance feedback models accordingly. Furthermore, query evaluation with our query engine is extremely fast, and working with large image collections is feasible (as opposed to experiments described in [9] with 286 images). MetaSeek from Columbia University [11] uses the relevance feedback model proposed by Rocchio, together with a normalization technique. CIRCUS [7] from EPFL, Switzerland, makes use of the user feedback to adopt the similarity metric. Similarity search in their system is based on Latent Semantic Indexing. The image retrieval system from the Computer Vision Group of the University of Geneva introduces the notion of feature frequency and adapts the concept of inverse document frequency to the image domain [12]. Finally, a system from the University of Bristol [16] uses an approach based on neural networks to interpret the relevance feedback.

We are aware of work on query models for similarity search and approaches to evaluate complex similarity queries. Fagin has proposed the A0-Algorithm that combines result lists from different subqueries and shows that it is optimal under certain middleware-specific assumptions that are somewhat restrictive [3, 15]. QuickCombine, an extension of the A0-Algorithm, significantly reduces the costs of merging result list but it still is much more expensive than our GeVAS-technique. Ciaccia et al. have come up with an evaluation scheme for complex queries with more than one reference object, but limited to one feature type [2].

5 Conclusions

Similarity search is an essential feature of image databases. The objective is to support the user such that his information need is covered in the best way possible. Relevance feedback gives the user control over the search process: the user explicitly states which results of the current step of the search are relevant, and which ones are not. The system uses this information to refine the query for the subsequent step. Several relevance feedback models, i.e., mappings of user feedback and the current query to a new, refined query, have been proposed. This article has given a review of these approaches.

Furthermore, we have described the relevance feedback component in our image retrieval system. It is realized as follows: we have an expressive query model for complex similarity search (e.g., several reference objects, explicit references to different features). Query evaluation is based on the VA-File [1] and is very efficient, even for large data sets. Our relevance feedback component maps the feedback and the current query to a new expression in the query language. We have explained that this is feasible for all relevance feedback models considered.

Future work includes comprehensive quality experiments on large data sets. The main problem hereby is the definition of a suitable test collection (like the TREC collections for text retrieval systems) and

feasible user tasks. E.g. looking for images of a certain person is only feasible if the features are able to discriminate between that person and other persons. Preliminary tests in our database have shown that relevance feedback significantly increases result quality with only a few steps.
Our retrieval systems currently contains more than 230,000 images, more than 20 feature types (mostly color and texture based), and all the feedback models described in this paper. The system is available online at `http://www-dbs.ethz.ch/~imageDB/`.

References

[1] K. Böhm, M. Mlivoncic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Roma, Italy, 2001.

[2] P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-Based Access Methods. In *Proceedings of the International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, p. 9–23, Valencia, Spain, 1998.

[3] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, p. 216–226, Montreal, Canada, 1996.

[4] R. Fagin and E. L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. In *Proceedings of the International Conference on Database Theory (ICDT)*, volume 1186 of *Lecture Notes in Computer Science*, p. 247–261, Delphi, Greece, 1997.

[5] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing Multi-Feature Queries for Image Databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, p. 419–428, Cairo, Egypt, 2000.

[6] B. Manjunath and W. Ma. Texture Features for Browsing and Retrieval of Image Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[7] Z. Pecenovic, M. Do, S. Ayer, and M. Vetterli. New methods for image retrieval. In *Proceedings of the International Congress on Imaging Science*, volume 2, p. 242–246, University of Antwerp, Belgium, 1998.

[8] J. Rocchio Jr. *Relevance Feedback in Information Retrieval, The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, p. 313–323. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.

[9] Y. Rui, T. Huang, and S. Mehrotra. Relevance Feedback Techniques in Interactive Content-Based Image Retrieval. In *Storage and Retrieval for Image and Video Databases (SPIE)*, p. 25–36, San Jose, California, USA, 1998.

[10] P. Schäuble. *Multimedia Information Retrieval, Content-based Information Retrieval from Large Text and Audio Databases*. Kluwer Academic Publishers, Zurich, Switzerland, 1997.

[11] J. Smith and S.-F. Chang. Searching for Images and Videos on the World-Wide Web. Technical Report 459-96-25, Columbia University, New York, New York, USA, 1997.

[12] D. Squire, W. Müller, and H. Müller. Relevance feedback and term weighting schemes for content-based image retrieval. In *The Third International Conference On Visual Information Systems*, p. 549–556, Amsterdam, The Netherlands, 1999.

[13] M. A. Stricker and A. Dimai. Color Indexing with Weak Spatial Constraints. In *Storage and Retrieval for Image and Video Databases (SPIE)*, volume 2670 of *SPIE Proceedings*, p. 29–40, San Diego/La Jolla, CA, USA, 1996.

[14] R. Weber, K. Böhm, and H.-J. Schek. Interactive-Time Similarity Search for Large Image Collections Using Parallel VA-File. In *Proceedings of the International Conference on Data Engineering (ICDE)*, p. 197, San Diego, CA, 2000.

[15] E. L. Wimmers, L. M. Haas, M. T. Roth, and C. Braendli. Using Fagin's Algorithm for Merging Ranked Results in Multimedia. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems*, p. 267–278, Edinburgh, Scotland, 1999.

[16] M. Wood, N. Campbell, and B. Thomas. Iterative Refinement by Relevance Feedback in Content-based Digital Image Retrieval. In *Proceedings of the 6th ACM international conference on Multimedia*, p. 13–20, Bristol, U.K., 1998.

Optimal Cache Memory Exploitation for Continuous Media: To Cache or to Prefetch?

Peter Triantafillou[1], Dept. of Computer Engineering, Tech. Univ. of Crete,

Antonis Hondroulis, Dept. of Computer Science, Stanford University,

Costas Harizakis, Dept. of Computer Engineering, Tech. Univ. of Crete.

Abstract

Network continuous-media applications are emerging with a great pace. Cache memories have long been recognized as a key resource (along with network bandwidth) whose intelligent exploitation can ensure high performance for such applications. Cache memories exist at the continuous-media servers and their proxy servers in the network. Within a server, cache memories exist in a hierarchy (at the host, the storage-devices, and at intermediate multi-device controllers). Our research is concerned with how to best exploit these resources in the context of continuous media servers and in particular, how to best exploit the available cache memories at the drive, the disk array controller, and the host levels. Our results determine under which circumstances and system configurations it is preferable to devote the available memory to traditional caching (a.k.a. "data sharing") techniques as opposed to prefetching techniques. In addition, we show how to configure the available memory for optimal performance and optimal cost. Our results show that prefetching techniques are preferable for small-size caches (such as those expected at the drive level). For very large caches (such as those employed at the host level) caching techniques are preferable. For intermediate cache sizes (such as those at multi-device controllers) a combination of both strategies should be employed.

1 Introduction

A fundamental issue when building the system support for continuous-media (CM) applications is how to best exploit the available cache memories, which nowadays exist at various levels in the system: at the (proxy) server's main memory, at intermediate multi-device controllers and at the embedded controllers of individual storage devices.

Problem Formulation and Overview of Contributions

The literature contains several strategies, which compete for memory and can be classified into two broad categories: caching and prefetching. A critical question is, therefore, whether the cache memory should be devoted for "consumption" by caching or by prefetching strategies. Under which circumstances and configurations is one preferable? And, how can we configure a cache so to achieve optimal performance? At optimal costs? Note that the size might be an important parameter, with different cache sizes calling for different solutions. Given that a real (proxy) server will contain caches of different size at the host level, at the disk array controller level, and at the drive level, the answers to these questions will prove of true interest to system designers. With this paper, we advocate solutions to these problems.

2 Related Work

2.1 Research in "Smart" Storage

Recently, there has been an increased interest for smart disk technology from both academia and industry [1,6,8,10,12]. These works propose new architectures and the migration of a great deal of processing into these smart devices. Our prior work in this area [15] differs in that it relies only slightly on the available embedded processors, specifically to perform tasks for carrying out directives for prefetching specific disk blocks. We center our attention on providing the necessary system support for continuous media applications. In general, the focus is on exploiting currently available resources of increasing capacity, i.e., the drive-level caches, to improve current performance and offer continuously greater performance benefits as this technology matures and develops (e.g., as the drive-level cache sizes increase).

[1] Contact author; email: peter@softnet.tuc.gr; URL: www.softnet.tuc.gr

2.2 Research in Caching Continuous Data

The basic idea behind the caching (or so-called "data sharing") techniques [5,7,13] is that if two clients, request the same video at different times, the server may service the latter one using the data, which is cached on behalf of the former one. Thus, the referenced video is read from disk only once, while the system can support in this way several simultaneous displays of the video.

2.3 Research in Prefetching Continuous Data into "Smart" Disk Caches

Nowadays drives with 8MB cache memory are common. In addition, while transfer rates increase fast, providing up to more than 15MB/sec, the positioning cost paid on every disk drive's seek improves at a much slower pace. Data prefetching is a policy, which minimizes the seek-to-transfer time ratio. Instead of reading a single disk data block more are fetched into a cache. Because of the CM data nature, soon the cached blocks will be also requested and thus two (or more) requests will have been served by "paying" only a single disk head positioning delay. Of course, the algorithms must ensure that the time to prefetch blocks of streams does not cause hiccups and/or high startup latencies. Several prefetching algorithms [15] have been proposed that significantly improve the maximum number of displays a device can support, at low latencies. We now briefly overview two techniques (presented in [15]) for prefetching video blocks which can improve I/O performance significantly.

2.3.1 Sweep & Prefetch algorithm (S&P)

Most related recent research has adopted the notion of scheduling with *rounds* [2,4,9,14,16,20]. Each continuous data object (stream) is divided into data segments such that the playback duration of each segment is some constant time (typically, from one to a few seconds). Within a round the storage server must retrieve from the disk the next data segment for all active displays, employing either a round-robin or a SCAN algorithm.

Scheme Sweep is the well-known continuous media retrieval scheme that reduces disk seek overhead to improve throughput. During a round, Sweep reads each stream's next segment with a SCAN like policy. Double buffers in main memory are needed to ensure continuous playback. The S&P [15] algorithm, briefly described below, employs prefetching as follows.

For a given duration of a round there is a specific upper bound in the number of streams N that can be supported by a disk. This number is mainly dependent on two parameters: the total positioning overhead (the time needed for the disk head to be positioned on the beginning of a segment) and the total transfer time (the time needed for the disk data to be transferred to main memory). Both of these delays are experienced when retrieving a data segment, which has not been prefetched. From now on this retrieval will be called "random retrieval". The number of data segments retrieved this way during a round (i.e., from the disk's surface), is denoted by the letter v. On the other hand, choosing to prefetch a data segment, incurs no additional positioning overhead. The number of prefetched segments in a round is denoted by the letter p.

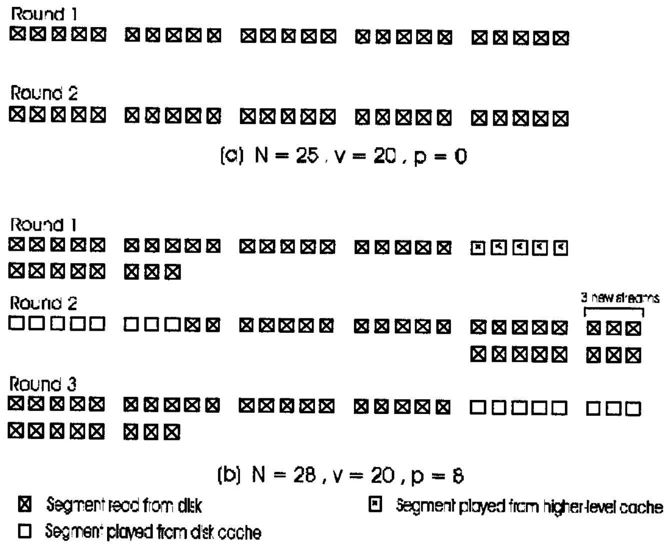

Round 1
Round 2
(a) N = 25, v = 20, p = 0
Round 1
Round 2
3 new streams
Round 3
(b) N = 28, v = 20, p = 8
Segment read from disk
Segment played from higher-level cache
Segment played from disk cache

Figure 1: Throughput increase via prefetching

For demonstration purposes of this series of schematic figures, we consider a disk drive with 10 MB/sec transfer time and a combined seek and rotational fixed overhead of 15 ms for every request. It is also assumed a constant display rate of 2 Mbps and round lengths of 1 sec. The latter implies that the size of any segment will be 250 KB. Given these values, the maximum number of segments that can be read within a round is 25, while the ratio of random retrieved segments that can be "exchanged" (in terms of time constraint) with prefetched segments, is 5 / 8 (40 ms and 25 ms retrieval times correspondingly).

During every round, v blocks are read from the surface and p blocks (one for each of the p streams) are prefetched. During the next round, those p segments will serve p streams at no positioning overhead. So there will be enough time to serve v additional streams and also prefetch the next segments from p of these v streams. Because the p segments are read at no positioning overhead, the total number of supported streams $(p + v)$ is greater. That is, the maximum throughput has been increased at the expense of exploiting cache memory, while the round and the size of the segments is kept constant.

Figure 1 demonstrates in a pictorial way the technique described. A box under another box stands for the next segment - contiguously placed on disk - of the same stream, while two adjacent boxes in the same line stand for the segments of two streams - that will be scheduled within a round in the same order as they appear. Figure 1(a) introduces the maximum number of streams (25 in this example) that can be supported with Sweep in a given configuration. Then, in Figure 1(b), the prefetching of 8 segments - one for each of 8 streams - is used to increase the supported number of streams from 25 to 28. Instead of randomly retrieving 25 segments, as scheme Sweep would do, in the S&P scheme there are 20 random retrievals. The time that scheme Sweep would spend to make five more retrievals, the S&P scheme uses it to prefetch 8 segments in each round. Note that the five segments that are 'neglected' by S&P are played from a higher-level cache. A straightforward way in order to avoid hiccups for these 'neglected' streams, is to delay these five streams enough to buffer an additional segment before their playback begins. In [15] the reader can find the details of the algorithms used so that the S&P algorithm can increase the maximum throughput without incurring high startup latencies and/or hiccups.

2.3.2 Group Periodic Multi-Round Prefetching algorithm (GPMP)

Scheme GPMP [15] introduces the concept of an epoch. The time interval of an epoch (or virtual round) is defined as the total duration of a fixed number $(u + 1)$ of actual rounds. The supported streams are grouped. There are $(u + 1)$ groups.

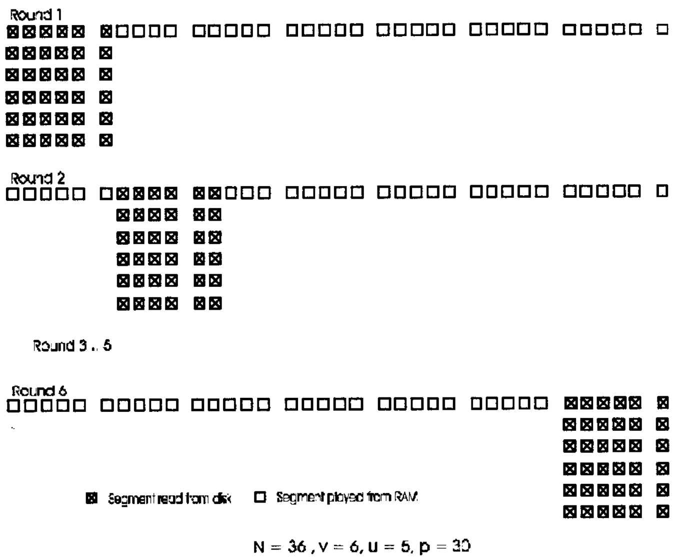


Figure 2: Grouped Periodic Multi-Round Prefetching (GPMP)

During a round, only v data segments will be retrieved from the disk, serving a group of v streams. There will also be enough time to prefetch u data segments for each one of these v streams. The letter u is used for the number of segments prefetched for the same video stream, while the letter p - used in the previous algorithm - was referring to the total number of prefetched segments.

The total streams supported under GPMP is $N = v * (u + 1)$. Figure 2 demonstrates GPMP in the same framework of figure 1. The maximum number of streams at $u = 5$ is 36 $(v = 6)$. In figure 2, as before, during Round 1 the 30 streams not served from the disk surface, have their segments already stored in the cache as a result of prefetching during the previous rounds. More specifically, as it is assumed that Round 1 corresponds to the status of an instance of the server working at its maximum throughput, each of the second group of 6 streams in the figure have 1 remaining segment in the cache, each of the third group of 6 streams have 2 remaining segments and so on. Recall that "played" segments are discarded immediately.

3 The Advocated Solution: Optimal Cache Utilization & Configuration

A simple way to combine both CM data caching and prefetching is to divide memory into two parts and optimally decide what percentage (including zero) to assign to each. An optimal way could mean maximizing the throughput (in displays per hour) or minimizing the average cost of serving a request (in $ per display). We will adapt a "scheduling" algorithm used differently in [13]. It is based on a system parameter called the *distance threshold* d_t. This parameter expresses the time distance threshold between two consecutive client requests for the same video, under which the latter will be served using data fetched earlier for the former – using data sharing techniques – and over which a new disk stream will be initiated – using prefetching techniques.

In fact, the value of the distance threshold d_t effectively partitions memory into 'caching' and 'prefetching' parts. The two limiting cases for the value of the distance threshold d_t are the prefetching only (d_t=0) and caching only ($d_t \rightarrow +\infty$). An answer to an optimality problem involves the estimation of the optimal value for the distance threshold d_t. Actually, a result such as d_t=0 implies that prefetching is the most beneficial technique, whereas $d_t \rightarrow +\infty$ implies that caching is preferred.

3.1 The 'Scheduling' Algorithm

As multiple streams must be supported concurrently, the media server typically serves them in *rounds* [2,4,9,14,16,20]. During a round of length R the system reads one segment from each stream that is adequate to sustain the playback of the stream for the duration of the round.

In the beginning of every round, for every new request arrival concerning the i^{th} video, the distance between that new request and its immediate leading request (*predecessor*) is calculated. The distance is measured in blocks of playback equal to the round length R. The distance between a pair of requests for the same video is considered 0 blocks if both requests arrive during the same round. Then, the scheduler decides whether to serve a new request by a new disk stream or by caching[2], based on the distance between the new request and its immediate predecessor for the i^{th} video. If the calculated distance is greater than d_t, then a new disk stream is scheduled to serve the new request. Else, the caching of the blocks for the predecessor request will also serve the new one. In the former case, when retrieving the video data from the disk, a prefetching technique (namely, the most efficient technique proposed in [15]) will be used to exploit available memory in order to improve performance. Thus, for every new request either the data sharing or the prefetching approach will be used.

3.2 Analysis and Experimentation Setup

Requests for videos arrive with mean rate λ. When in steady state, the service rate (throughput) μ equals the arrival rate λ. The inter-arrival time distances are considered independent and identically distributed random variables. An exponential distribution is considered, with density $f(x) = \lambda e^{-\lambda x}, x \geq 0$ and (cumulative) distribution function $F(x) = 1 - e^{-\lambda x}$, $x \geq 0$. The total number of videos is V and each has popularity p_i, $1 \leq i \leq V$ and playback duration L. It is assumed that the requests are served in rounds of duration R. Also, a specific distance threshold d_t is given.

In summary:

SYMBOL	DESCRIPTION
λ	Poisson request arrival **mean rate** (reqs/sec)
V	Total **number of** available **videos**
p_i ,$1 \leq i \leq V$	**Popularity** (access) distribution
L	**Video duration** (secs)
R	**Round duration** (secs)
Crate	Video **consumption rate** (in KBytes per sec)
d_t	Distance threshold
CostPerDisk	Cost of a single disk (in $)[3]
CostPerMByte	Cost of 1 Mbyte of memory (in $)

The next table contains a list of symbols that will be used throughout the following analysis:

[2] Even in the case of caching, a temporary new I/O stream for the new request is necessary, until playback reaches the point where the predecessor was when the request arrived.

[3] This is the "disk bandwidth" cost for a single disk unit: the cost of the imbedded disk-cache has been deducted.

SYMBOL	DESCRIPTION
N_{tot}	Total **concurrent displays**
N_s	Concurrent **displays** supported **by caching**
N_p	Concurrent **disk streams** (one per display) supported **by**
Mem	**Total memory** available (in Kbytes)
Cmem	**'Caching' memory** requirements (in Kbytes)
Pmem	**'Prefetching' memory** requirements (Kbytes)
p_s	Probability of caching (for consecutive requests)
q_s	Probability of prefetching (for consecutive requests)
$Cost_{tot}$	Configuration **cost** (in $)
Disks	**Total disks** available
MaxDiskThr	Max 1 disk throughput (in concurrent streams)

The parameters included in the second table, when subscripted refer to the i^{th} video. The parameters N_{tot}, N_s and N_p can be considered the throughput, the "caching" throughput and the "prefetching" throughput of the system respectively, in displays per time unit (which is equal to L).

Our goal is to derive the $\lambda_{\max}$ value and the $d_{t,\,opt}$ value (that yields this maximum throughput $\lambda_{\max}$), as they depend on the basic system and problem parameters.

3.3 Analysis

In the analysis we refer to "server" and "cache" without particular mention to host, disk array controller or disk with their respective caches.

3.3.1 Estimating data sharing cache size & I/O stream requirements

Given the parameters included in the 1st table, the following is an estimation of N_p and CMem requirements. It can be shown that the requests for the i^{th} video, are separated by inter-arrival time distances that are also i.i.d. random variables, exponentially distributed with mean arrival rate:

$$\lambda_i = p_i \cdot \lambda$$

The server model assumed is an M/D/∞ model, with constant service time L. Schematically:

$$\begin{tikzcd} {} \arrow[r, "\lambda_i"] & \boxed{\text{M/D/}\infty} \arrow[r] & {} \end{tikzcd}$$

The number of requests for the i^{th} video (state of the system) N_i ,$1 \le i \le V$, follows the Poisson distribution [17,18], so its mean and variance are:

$$N_i = \lambda_i \cdot L \qquad\qquad \sigma^2_{N_i} = \lambda_i \cdot L$$

The total number of requests N_{tot} served by the system has the following mean and variance:

$$N_{tot} = \sum_{i=1}^{V} N_i = \lambda \cdot L \qquad\qquad \sigma^2_{N_{tot}} = \sum_{i=1}^{V} \sigma^2_{N_i} = \lambda \cdot L$$

We call a pair of consecutive requests for the same video a *sharing pair* of that video if the distance between those requests is not greater than d_t. The lagging request in a sharing pair can be served by caching. A new disk stream must be used to serve the lagging request if it and the leading request form a *non-sharing pair*. The probabilities of a sharing pair and a non-sharing pair for the i^{th} video are:

$$p_{s_i} = 1 - e^{-\lambda_i \cdot (d_t+1) \cdot R} \qquad\qquad q_{s_i} = 1 - p_{s_i} = e^{-\lambda_i \cdot (d_t+1) \cdot R}$$

Consequently, the disk streams requirement for the i^{th} video, $1 \le i \le V$, are:

$$N_{p_i} = q_{s_i} \cdot N_i = \lambda_i \cdot q_{s_i} \cdot L \qquad\qquad \sigma^2_{N_{p_i}} = q^2_{s_i} \cdot \sigma^2_{N_i} = \lambda_i \cdot q^2_{s_i} \cdot L$$

And thus, the mean and variance of the total concurrent disk streams N_p are:

$$N_p = \sum_{i=1}^{V} N_{p_i} = L \cdot \sum_{i=1}^{V} \lambda_i \cdot q_{s_i} = L \cdot \lambda \cdot \sum_{i=1}^{V} p_i \cdot e^{-\lambda \cdot p_i \cdot (d_t+1) \cdot R} \quad , \ (1)$$

$$\sigma^2_{N_p} = \sum_{i=1}^{V} \sigma^2_{N_{p_i}} = L \cdot \sum_{i=1}^{V} \lambda_i \cdot q^2_{s_i} = L \cdot \lambda \cdot \sum_{i=1}^{V} p_i \cdot e^{-2 \cdot \lambda \cdot p_i \cdot (d_t+1) \cdot R} \quad , \ (2)$$

The number of displays for the i^{th} video N_{s_i} being served by caching has the following mean and variance:

$$N_{s_i} = p_{s_i} \cdot N_i = \lambda_i \cdot p_{s_i} \cdot L \qquad\qquad \sigma^2_{N_{s_i}} = p^2_{s_i} \cdot \sigma^2_{N_i} = \lambda_i \cdot p^2_{s_i} \cdot L$$

If random variable Δ_i is the distance (in blocks of duration R) between the leading and the lagging request in a pair for the i^{th} video, sharing or non-sharing, then:

$$P\{\Delta_i = k\} = P\{\Delta_i \le k \cap \overline{\Delta_i \le k-1}\} = (1-e^{-\lambda_i \cdot (k+1) \cdot R}) - (1-e^{-\lambda_i \cdot k \cdot R}) = e^{-\lambda_i \cdot k \cdot R} \cdot (1-e^{-\lambda_i \cdot R})$$

The distribution of the distance Δ_{s_i} between a sharing pair for the i^{th} video is:

$$P\{\Delta_{s_i} = k\} = P\{\Delta_i = k \mid \Delta_i \le d_t\} = \frac{P\{\Delta_i = k \cap \Delta_i \le d_t\}}{P\{\Delta_i \le d_t\}} = \frac{P\{\Delta_i = k\}}{P\{\Delta_i \le d_t\}} = \frac{e^{-\lambda_i \cdot k \cdot R} \cdot (1-e^{-\lambda_i \cdot R})}{1-e^{-\lambda_i \cdot (d_t+1) \cdot R}} \quad ,$$

$$\mathrm{k} = 0, 1, \ldots, d_t$$

and therefore its mean and variance are:

$$E\{\Delta_{s_i}\} = E\{\Delta_i \mid \Delta_i \le d_t\} = \sum_{k=0}^{d_t} k \cdot P\{\Delta_{s_i} = k\} = \frac{1-e^{-\lambda_i \cdot R}}{1-e^{-\lambda_i \cdot (d_t+1) \cdot R}} \cdot \sum_{k=0}^{d_t} k \cdot e^{-\lambda_i \cdot k \cdot R}$$

$$Var\{\Delta_{s_i}\} = \sum_{k=0}^{d_t} \left(k - E\{\Delta_{s_i}\}\right)^2 \cdot P\{\Delta_{s_i} = k\}$$

Let $D_{k,i}$ denote the distance between the k^{th} sharing pair for the i^{th} video. The random variables of the sequence of distances $D_{k,i}$ are i.i.d. with distribution that of Δ_{s_i}. The caching memory requirement for the i^{th} video is:

$$CMem_i = CRate \cdot R \cdot \sum_{k=1}^{N_{s_i}} D_{k,i}$$

Hence, assuming[4] independence of N_{s_i} and the sequence of $D_{k,i}$, its mean and variance[5] are:

$$CMem_i = R \cdot CRate \cdot N_{s_i} \cdot E\{\Delta_{s_i}\}$$

$$\sigma^2_{CMem_i} = (R \cdot CRate)^2 \cdot \left(E^2\{\Delta_{s_i}\} \cdot \sigma^2_{N_{s_i}} + Var\{\Delta_{s_i}\} \cdot N_{s_i}\right)$$

Therefore, the following are the mean and variance of the required 'caching' memory[6]:

$$\boxed{\begin{gathered} CMem = \sum_{i=1}^{V} CMem_i = CRate \cdot R \cdot \sum_{i=1}^{V} \left(N_{s_i} \cdot E\{\Delta_{s_i}\}\right) = CRate \cdot R \cdot \lambda \cdot L \cdot \sum_{i=1}^{V} p_i \cdot \left(1-e^{-\lambda \cdot p_i \cdot R}\right) \cdot \left(\sum_{k=0}^{d_t} k\, e^{-\lambda \cdot p_i \cdot k \cdot R}\right), (3) \\ \sigma^2_{CMem} = (R \cdot CRate)^2 \cdot \sum_{i=1}^{V} \left(E^2\{\Delta_{s_i}\} \cdot \sigma^2_{N_{s_i}} + Var\{\Delta_{s_i}\} \cdot N_{s_i}\right) \quad , \quad (4) \end{gathered}}$$

3.3.2 Optimal memory utilization (to cache or to prefetch)

Given Disks, MaxDiskThr and Mem from the second table and all parameters of the first table, excluding λ and d_t, in order to maximize the server throughput μ = λ (in concurrent displays per second), the optimal values for d_t, CMem and PMem are calculated using the following formulae:

$$d_{t,opt} = \arg\max_{d_t} \{\lambda \mid (N_p(\lambda, d_t) \le Disks \cdot MaxDiskThr) \cap (CMem(\lambda, d_t) + PMem(N_p(\lambda, d_t)) < Mem)\}$$

$$\lambda_{\max} = \max\{\lambda \mid (N_p(\lambda, d_{t,opt}) \le Disks \cdot MaxDiskThr) \cap (CMem(\lambda, d_{t,opt}) + PMem(N_p(\lambda, d_{t,opt})) < Mem)\}$$

$$\boxed{\begin{gathered} CMem_{opt} = CMem(\lambda_{\max}, d_{t,opt}) \quad , \quad (5) \\ PMem_{opt} = PMem(N_p(\lambda_{\max}, d_{t,opt})) \quad , \quad (6) \end{gathered}}$$

[4] Despite the fact that the random variables N_{s_i} and $D_{k,i}$ are correlated, and the fact that detailed analysis becomes really complex, the results even with this simplifying assumption matches very well experimental data

[5] A detailed analysis of that can be found in [19], p.183

[6] The maximum caching memory requirement for the i^{th} video is $L \cdot CRate$, i.e. the whole video

Determination of the above four variables can be easily implemented by use of a lookup table containing pre-calculated values of $N_p(\lambda, d_t)$ and $CMem(\lambda, d_t)$, the pre-calculation which uses formulae (1) and (3).

3.3.3 Optimal server configuration – Minimizing its cost

In a very similar way, we can estimate the minimum cost (in \$ per display) of a video server capable of supporting a given throughput $\mu = \lambda$ (in concurrent displays per sec). Assuming MaxDiskThr[7], CostPerDisk and CostPerMByte from the second table and all parameters of the first table except d_t, the optimal values for d_t, Disks and Mem are calculated using the following formulae (here, $N_p(\lambda_{given}, d_t)$ and $CMem(\lambda_{given}, d_t)$ are considered functions of d_t only):

$$d_{t,opt} = \arg\min_{d_t} \left\{ \begin{array}{l} Disks \cdot CostPerDisk + \\ \{CMem(d_t) + PMem(N_p(d_t))\} \cdot \frac{CostPerMByte}{1024} \end{array} \middle| Disks \geq 1 \cap N_p(d_t) \leq Disks \cdot MaxDiskThr \right\}$$

$$Disks_{opt} = \arg\min_{Disks} \left\{ Disks \cdot CostPerDisk + \{CMem(d_{t,opt}) + PMem(N_p(d_{t,opt}))\} \cdot \frac{CostPerMByte}{1024} \right\} \quad , (7)$$

$$Mem_{opt} = CMem(d_{t,opt}) + PMem(N_p(d_{t,opt})) \quad , (8)$$

Again, determination of the above two variables can be easily implemented by use of a lookup table containing pre-calculated values of $N_p(\lambda, d_t)$ and $CMem(\lambda, d_t)$. In fact, only $N_p(\lambda_{given}, d_t)$ and $CMem(\lambda_{given}, d_t)$ are necessary for giving an answer to the current problem.

4 Validation and performance results

The effectiveness of the proposed solutions has been studied extensively by simulating a server that uses the aforementioned scheduling algorithm and serves every incoming request for display without any delays.

We have simulated a server that delivers videos with a delivery rate of 250 KB/sec. Movies are assumed to be one hour long so that the total amount of storage required per movie is approximately 880 Mbytes. Disks have an average transfer rate of 14MB/sec, 10 zones, an average seek delay of 5ms, 10,000 rpm, and 10GB storage capacity. The disk serves incoming requests using the standard elevator/SCAN scheduling algorithm while the seek cost model used is the standard one [21]. Arrivals follow a Poisson process, with mean rate λ. Simulations were ran with λ in the range from 1 req/hr to 3,558 req/hr (400 - 800 are some typical values [3] for active clients for VoD servers). For simplicity, powers of 2 have been used for values of parameter d_t, from 1 and up to 4096. The values used for the round R are 0.25 sec, 0.5 sec and 1.0 sec. Video popularity follows a Zipfian distribution with distribution parameter $\theta = 0.271$ as is typically done [3].

The cache memory is modeled as a collection of blocks of size equal to $R \cdot CRate$, ($CRate$ is the video "consumption" rate). Out simulations have durations over 250,000 hrs, which are adequate for the computation with sufficient confidence of the average values and variances for the number of concurrent displays, the average concurrent disk streams, and the average caching-memory requirements.

4.1 Validation and Results on Optimal Cache Utilization

We first measure the mean and variance of both N_p (number of I/O streams for which the prefetching policy is used) and CMem (the memory required for data sharing).

Figure 3 illustrates the optimal memory partitioning. Also, illustrated in figure 4 are the corresponding mean and max concurrent disk I/O streams during optimal performance (maximum server throughput), along with the maximum possible concurrent I/O streams the disk can offer for the respective available memory, if that memory were dedicated to prefetching.

It is evident by figure 4 that during optimal performance, a server with as little memory as 128 Mbytes, serves requests using prefetching and takes the most out of that technique for the available memory. Of course, if still there is an unused portion of memory left, that portion is given to the caching approach, since it can offer some (although limited) extra throughput. In fact, this explains the anomaly shown in figure 3, for 32 Mbytes of available memory. It is clear from figure 4 that with the available memory being equal to 32 Mbytes, the maximum concurrent disk streams instantly used by the server are nearly 38. To support those streams our analysis and experimentation shows that 23 Mbytes are necessary.

[7] MaxDiskThr can be set to a smaller value, in order to account for some specific disk utilization other than that assumed during the disk's peek performance.

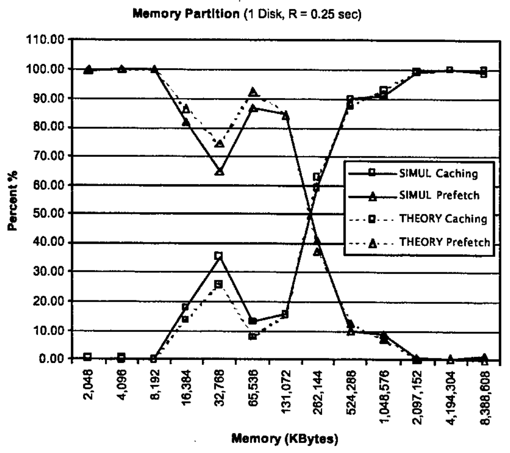


Figure 3

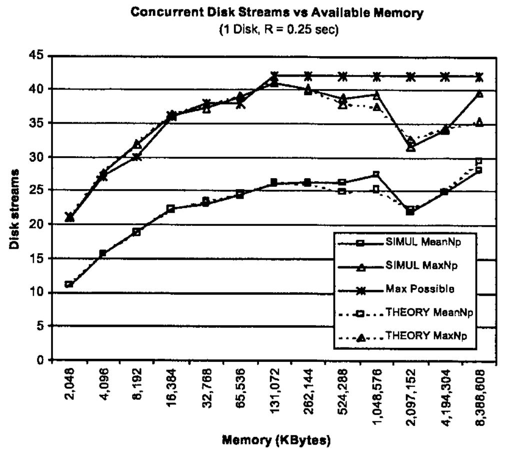


Figure 4

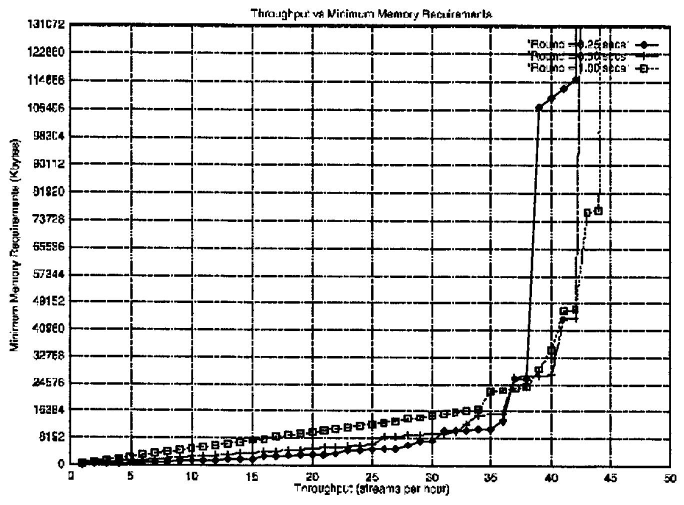


Figure 5

Figure 5 shows the memory requirements in order to achieve the desired throughput using prefetching only techniques, given the round duration R. The server cannot use more that 38 concurrent streams because this would require at least 100Mbytes as shown in figure 5. Thus, it only uses 23 Mbytes for prefetching, leaving the other 9 Mbytes for caching. Still, prefetching is more efficient in those ranges of available memory, that as soon as the necessary 100 Mbytes are available, e.g. when there are 128 Mbytes of memory, memory is again almost completely devoted to prefetching. That of course explains why the curves in figure 3 go up and then down near 32 Mbytes.

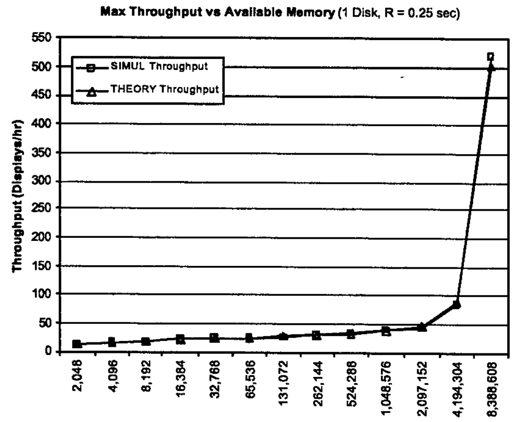


Figure 6

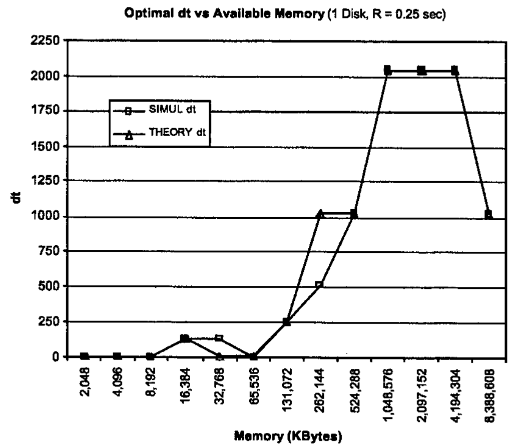


Figure 7

Of course, when the upper bound on concurrent disk streams is reached, all remaining memory is given to caching. It is interesting to note that as the available memory increases, caching is becoming more effective than prefetching. That is why in figure 4 there is a decrease in both the mean and maximum concurrent disk streams the server uses for larger caches.

Last, figures 6 and 7 show the values for $\lambda_{\max}$ and $d_{t,opt}$ corresponding to the optimal performance. Certain mismatches (peeks) are due to the selected coarse granularity for d_t.

4.2 Results on Optimal Server Configuration for Minimal Cost

A comparison of the theoretically estimated and measured minimal server cost is performed. The total cost is the sum of the cost for disks ($Disks \cdot CostperDisk$) and the cost of the total memory ($(CMem + PMem) \cdot \frac{CostperMByte}{1024}$). Also, the values used for Disks and $Mem = CMem + PMem$ are thus found. *Cmem and Pmem* refer to the cache space required for caching and prefetching.

Figures 8 and 9 show the optimal cost server in terms of Disks and Mem, for different values of throughput. The memory requirements are increasing with increasing server throughput, as expected.

Similarly, the required number of disks increases as the throughput requirements increase. Interestingly enough, for R = 0.25 in figure 9, there is a decrease from 8 to 7 disks required for optimal configuration, when throughput increases from 400 to 800 displays/hr. This is because between 400 and 800 displays/hr we have already overcome the crucial point of required memory, beyond which caching is starting to be more efficient than prefetching. Thus at that high throughput the cost of increasing throughput by caching is cheaper (incurs less increase in $ for extra caching memory) than that of increasing it by prefetching. Actually, at 800 displays/hr it is even cheaper to replace a whole disk with caching memory. This is exactly analogous to what happens in figure 4 when the available memory increases beyond 1 Gbyte. A similar drop happens, for the cases of R being equal to 0.5 and 1 secs, when the server throughput is further increased (although not shown here).

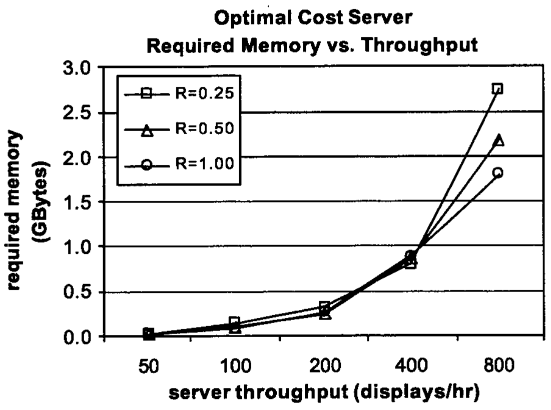


Figure 8

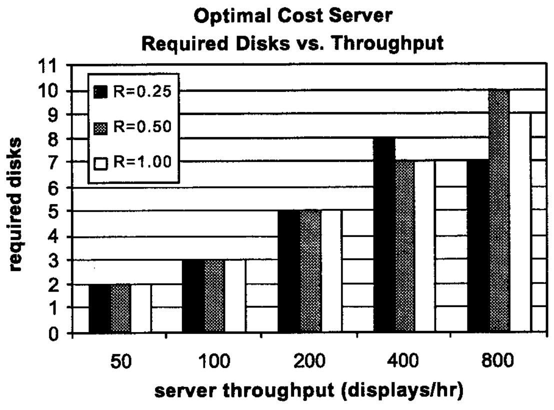


Figure 9

Figures 8 and 9 show that with increasing R the peek value for required disks is increasing. That means that as R increases it becomes more and more cost-effective to further utilize prefetching.

5 Contributions and Concluding Remarks

Network continuous-media applications are emerging with a great pace. Cache memories have long been recognized as a key resource (along with network bandwidth) whose intelligent exploitation can ensure high performance for such applications. Cache memories exist at the CM servers and their proxy servers in the network. Within a server, cache memories exist in a hierarchy (at the host, the storage-devices and at intermediate multi-device controllers).

Our position is that it is of fundamental importance therefore, for the efficient system support for such applications, to devise optimal cache exploitation strategies. Especially in the light of the large number of memory exploitation strategies which have been proposed by researchers, and the fact that these strategies compete against each other for memory resources, it is important to optimally utilize and configure caches at all levels they appear.

Our advocated solution shows that prefetching is clearly more advantageous than caching, if the available memory is up to 128 Mbytes, regardless of round R. Considering the expected size for on-board disk caches, which is 64 Mbytes (prediction for year 2001/2), prefetching is the desirable approach at the disk level. The same result also reveals that for available memory beyond 1 Gbyte caching is significantly more effective. So for the host level caches, which are expected to be at least a few Gbytes, the traditional CM data sharing techniques offer indeed the best way to exploit this memory. It should be stressed that for intermediate level caches found in a typical hierarchical server (e.g., a cache of 200MB at the disk-array controller) our results advocate the employment of the indicated optimal mixture of CM data sharing and prefetching techniques to exploit the available cache.

Our goal is to aid in the configuration of a modern CM system. To this end:

1) Given the basic parameters like the mean request arrival rate λ, the number of videos V, the access distribution etc., using the derived formulas (1)-(4) one can determine the memory requirements for caching and prefetching purposes.
2) Given the available memory, formulas (5)-(6) can be used to determine the optimal way to partition the memory into caching and prefetching partitions so as to maximize the achievable throughput.

Given a requirement on the achievable throughput, using formulas (7) and (8) one can determine the cheapest solution for the video server, in terms of the number of disks and the amount of necessary memory (including the best way to partition this memory).

6 References

[1] D. Anderson, Network Attached Storage Research, talk given in the spring 1998 NSIC/NASD Workshop, (available from http://www.nsic.org/nasd)

[2] S. Berson, S. Ghandeharizadeh, R.R. Muntz and X. Ju. Staggered Striping in Multimedia Information Systems. Proc. of the Intern. Conf. on Management of Data (SIGMOD), Minneapolis, Minnesota, pp. 79-90, 1994.

[3] Asit Dan, Dinkar Sitaram, Buffer Management Policy for an On-Demand Video Server, IBM Research Report RC 19347, Oct. 1998

[4] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. Multimedia Storage Servers: A Tutorial. IEEE Computer, Vol.28, no.5, May 1995, pp.40-49.

[5] L. Golubchik, J.C.S. Lui and R.R. Muntz, Adaptive Piggybacking: A novel technique for data sharing in video-on-demand storage servers, Multimedia Systems (1994) 4: 140-155

[6] J. Gray, Put everything in the storage device, Talk given in the spring 1998 NSIC/NASD Workshop, (available from http://researsh.microsoft.com/~gray)

[7] M. Kamath, K. Ramamritham and D. Towsley, Buffer Management for Continuous Media Sharing in Multimedia Database Systems, Technical report 94-11 University of Massachusetts 1994

[8] K. Keeton, D. Patterson and J. Hellerstein, A Case for Intelligent Disks (IDISKs), SIGMOD Record, Vol. 27, Number 3, August 1998

[9] B. Ozden, R. Rastogi and A. Silberschatz. Disk Striping in Video Server Environments. In Proc. of the Intern. Conf. on Multimedia Computing and Systems (ICMCS), June 1996

[10] D. Patterson and K. Keeton, Hardware Technology Trends and Database Opportunities, Keynote address at SIGMOD '98, June 1998

[11] A.L.N. Reddy and J.C. Wyllie. I/O Issues in a Multimedia System. IEEE Computer, March 1994, pp. 69-74

[12] E. Riedel, G. Gibson and C. Faloutsos, Active Storage for Large-Scale Data Mining and Multimedia Applications, In Proc. Of Int. Conf, on VLDB, 1998

[13] Weifeng Shi, Data sharing in interactive continuous media servers. Ph.D. dissertation, U.Southern California, Sept. 1998

[14] P. Triantafillou and C. Faloutsos. Overlay Striping and Optimal Parallel I/O in Modern Applications. Parallel Computing, January 1998, (24) pp 21-43.

[15] P. Triantafillou and S. Harizopoulos, Prefetching into smart disk caches for high performance video servers, IEEE Int. Conf. on Multimedia Computing and Systems, June 1999. (See also IEEE Concurrency, July 2000).

[16] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. Multimedia Storage Servers: A Tutorial. IEEE Computer, May 1995, pp.40-49.

[17] D. Gross, C. Harris, Fundamentals of queuing theory, 1985

[18] L. Kleinrock, Queuing systems, 1975

[19] A. Papoulis, Probability, random variables and stochastic processes, 1984

[20] S. Ghandeharizadeh, S.H. Kim and C. Shahabi. On Disk Scheduling and Data Placement for Video Servers. ACM Multimedia Systems, 1996.

[21] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. IEEE Computer, March 1994, pp. 17-28.

An Architecture for Redundant Multicast Transmission Supporting Adaptive QoS

Ch. Bouras[1,2] **A. Gkamas**[1,2] **An. Karaliotas**[1,2] **K. Stamos**[1,2]

[1]***Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece***

[2]***Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece***

e-mail: {bouras, gkamas, karaliot, stamos}@cti.gr

Abstract

In this paper we describe the architecture and the implementation of an application that was developed for the transmission of multimedia data, using the multicast mechanism, over the Internet. There are two major issues that have to be considered when designing and implementing such a service, the fairness and the adaptation schemes. The fairness problem results from the fact that receivers with different capabilities have to be served. In our application we use a mechanism that categorizes the receivers into a number of groups according to each receiver's capabilities and (the mechanism) serves each group of users with a different multicast stream. With the term "capabilities" we do not only mean the processing power of the client, but also the capacity and the condition of the network path towards that client. Because of today's Internet heterogeneity and the lack of Quality of Service (QoS) support, the sender cannot assume that the receivers will permanently be able to handle a specific bit rate. We have therefore implemented an additional mechanism for the intra-stream bit rate adaptation.

Keywords: Networking protocols and media delivery, Multimedia distribution and transport, Multicast, Quality of Service, Adaptation mechanisms, CORBA

1 Introduction – Related Work

The heterogeneous network environment that Internet provides to the real time applications as well as the lack of sufficient Quality of Service (QoS) guarantees, many times forces applications to embody adaptation elements in order to work efficiently. The main goal of such an approach is to adapt the data rate that is sent to the network every time that network conditions change. The decision whether the rate will increase or decrease is based on feedback information that the receivers send back to the sender. Many researchers believe that this end-to-end control scheme must be implemented in the application layer because today's Internet architecture does not provide such a mechanism in the network layer ([20], [13]).

The implementation of adaptation mechanisms in the applications is often criticized. The main arguments that rise against it are that the technologies that are used today for the implementation of the core networks, typically based on ATM (Asynchronous Transfer Mode) technology, provide capabilities to support QoS; as a result the network should offer to the applications QoS guarantees. This is generally true but there is a big problem about it: Today's Internet is divided into thousands of different administration domains. The QoS strategies that are implemented on each one are certainly different, and in many cases no QoS strategy is implemented at all. So the multimedia data flows that have to traverse many of these different domains in order to reach to the end user don't have a sufficient QoS support.

In addition any application that sends data (mostly multimedia) over the Internet should have a friendly behaviour towards the other flows that coexist in today's Internet and especially towards the TCP flows that comprise the majority of flows. Due to the sliding window algorithm that TCP deploys, such flows are the most sensitive in network congestion conditions. Therefore the sliding

window algorithm forces applications to meet some special characteristics in order to be characterized as TCP friendly ([26]).
The system we propose is based on multicast video transmission with the use of RTP/RTCP ([10]). The main perspectives we tried to fulfil are 1) each receiver should receive the best video quality that it is capable of and 2) the generated multicast data flow should not be a constraint for the other flows.
In order to achieve the first goal we create *n* different streams (in most network conditions a small number of different streams is enough -typically 3 or 4 streams), each one within certain bandwidth limits. All the streams carry the same video information, each one of them having a different quality. Receivers join in the appropriate stream depending on the condition of the network path towards them and the processing power of each one. If meanwhile the receiver detects that the stream it has joined isn't suitable for it any more another implemented mechanism is used in order to provide the receivers the capability of moving into another stream.
In order to achieve the second goal, we deploy the Additive Increase Multiple Decrease (AIMD) scheme in the inter-stream adaptation algorithm. The adaptation mechanism adapts the rate of each stream taking into account the number of the receivers that are congested or unloaded. In addition, if the capabilities of a receiver aren't suitable for the stream it has joined, it moves to another stream with lower or higher bandwidth limits.
The subject of adaptive streaming of multimedia data over networks has engaged researchers all over the world. The simplest approach for the sender is to use a unicast connection towards each receiver ([1], [2], [3], [5], [6], [14]). This approach is best adapted to each receiver's receiving capability, but has the drawback of making unnecessary use of network resources and cannot therefore scale well into a large number of receivers. In order to overcome the above drawback, someone must use multicast transmission ([23]). The methods proposed for the multicast transmission of time sensitive data in the Internet can be generally divided in three main categories, depending on the number of multicast sessions used:

1. The source uses a single multicast session for all receivers ([4], [16],[15], [27]). This results to the most effective use of the network resources, but on the other hand the fairness problem arises.
2. Simulcast: The source transmits versions of the same video encoded in varying degrees of quality. This results to the creation of a small number of multicast sessions with different rates, responsible for a range of receivers with similar capabilities ([17], [21], [23]). The different streams carry the same video information but in each one the video is encoded with different bit rates, and even different video formats (MPEG, H263, JPEG). So each receiver joins in the session that carries the video quality, in terms of bit rate, that is capable of receiving. The main disadvantage in this case is that the same video information is replicated over the network.
3. The source uses layered encoded video, which is video that can be reconstructed from a number of discrete data streams and transmit each layer into different multicast session ([18], [19]). The receivers subscribe to one or more multicast sessions depending on the available bandwidth into the network path to the source. The video is divided in to one basic stream and more additional streams. The basic stream provides the basic quality and the quality improves with each layer added.

This work is based on the simulcast approach and it is an extension of the work, which has been presented in [24] and [23]. The rest of this paper is organised as follows: Section 2 presents the architecture of the implemented prototype. In section 3, we give a detailed description of the operation of our prototype application. Section 4 presents some implementation issues. In section 5, we present some initial results in performance evaluation of the implemented prototype. Finally, section 6 concludes the paper and discusses some of our future work.

2 System Architecture

Our system is based on the multicast transmission of video. The advantages that the IP multicast service has, makes it the only choice for transmitting multimedia data, especially when the

application is emulating some kind of broadcasting over the Internet. On the session layer, according to the OSI reference model, the Real Time Protocol (RTP) - Real Time Control Protocol (RTCP) - is used both by H.323 application and MBONE and is broadly considered as the de facto standard for multimedia transmission over the Internet. The RTP-RTCP protocol was also used because of the feedback capabilities that it offers (RTCP reports). At the same time, a unicast session is established between each client and the Server. This session provides the necessary information to the client concerning the multicast IP address as well as the port that is used. So the main multicast data session can be joined. Because of the variations on the quality of video that various clients can handle, the source transmits a small number of (in our implemented prototype we use three) different multicast video streams, each one with its own bandwidth limits, with no overlapping. The transmission rate within each stream is adapting within its limits according to the capabilities and the state of the clients participating in.
The main goals that we tried to fulfil are: 1) A quite flexible and fast application, which can react fast to the network changes, 2) adjust its rate in such a way that keeps a friendly behaviour against other TCP or UDP network applications, and 3) achieve the best possible fairness for each receiver based on the idea that fairness for each receiver means to give a slightly worse video quality compared to that the receiver might have if it was alone in the multicast stream. The system architecture consists of two major entities: The server and the client.

2.1 Server Architecture

The server is unique and responsible: 1) to create of the *n* different multicast streams, 2) to set each one's bandwidth limits, 3) to track if there are any clients that are not handled with fairness and 4) to provide the mechanisms to the clients to change stream whenever they consider that they should be in another stream closer to their capabilities. Before the beginning of the transmission the Server entity interacts with the system administrator, with a quite simple graphical user interface, in order for the user to set the desirable values to the main variables of the application. These variables are the number of the streams that will be created, the bandwidth limits of each one, the multicast IP address and the ports, data and control and the video that will be transmitted.

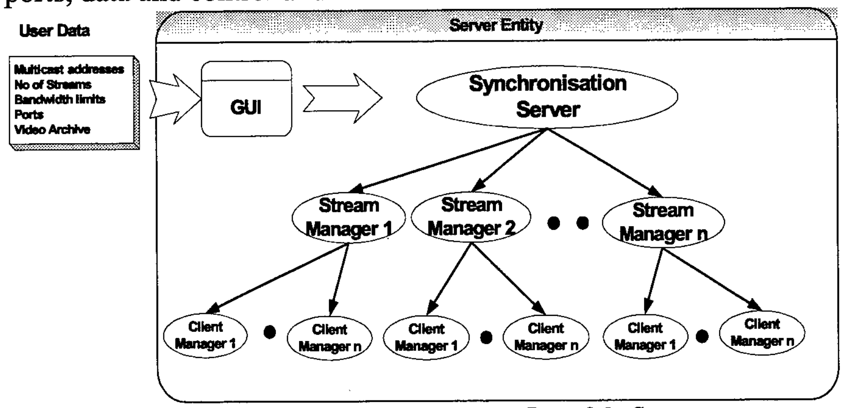


Figure 1 The architecture and the data flow of the Server.

In the above figure the organisation and the architecture of the Server entity is shown. The Server generates *n* different Stream Managers. In each Stream Manager an arbitrary number of Client Managers is assigned. Each Client Manager corresponds to a unique receiver that has joined the stream controlled by this Stream Manager.
The server architecture can be further decomposed in the following modules:

- **Synchronisation Server**: Responsible for the management, synchronization and intercommunication between Stream Managers. When a Stream Manager wants to resume the video transmission that was stopped because either all of its clients have left towards another stream or have stopped receiving video, it has to know where the transmission of the video from the other streams currently is. It therefore requests this information from the Synchronisation Server.

- **Video archive:** The hard disks where the video files are stored. Our implementation supports three video formats, namely H.263, MPEG and JPEG.
- **Stream Manager:** The Stream Manager entity is created by the server entity. It is responsible for the maintenance and the monitoring of one of the *n* different multicast streams that are generated in the beginning of the application. Also the Stream Manager entity has all the intra-stream adaptation mechanisms for the adjustment of the transmission rate. These mechanisms will be described later on. A Stream Manager uses the information provided by Client Managers to adjust its data rate in order to achieve optimum overall results. A Stream Manager can only transmit when it has at least one client participating in its multicast session. The Stream Manager entity contains the following modules.
  - <u>Optimal Rate Estimation</u>: The Feedback Analysis module (described later in detail) contained in every Client Manager processes the RTCP reports from the Client and declares the Client to be in one of three states: CONGESTED, LOADED or UNLOADED. The Optimal Rate Estimation module periodically gathers the states reported by all Client Managers belonging to its Stream Manager at the end of a specific, fixed time period (from now on called an epoch). It then uses an algorithm described in a following paragraph that tries to improve fairness between clients by determining whether a lower or a higher bit rate is more appropriate. Whenever a client cannot be satisfied by a stream due to the fact that most of the other receivers have much higher or much lower reception capabilities, the Optimal Rate Estimation module informs it that it has to move to a lower or higher quality stream.
  - <u>Quality Adaptation</u>: Responsible for the adaptation of the video transmitting quality. Its behaviour (whether to adapt the transmitting quality upwards or downwards) is determined by the Optimal Rate Estimation module. Due to the fact that the new bit rate depends on the value of the current one, the Quality Adaptation module computes the current bit rate by dividing the amount of bytes sent from the end of the last epoch until the end of this epoch, by the time period that is represented by an epoch. The other alternative would be to make no manual computation and to assume that the current bit rate is the one that the module tried to establish at the end of the previous epoch. This second approach has the decisive drawback however, that there is no guarantee that the previously intended bit rate was actually achieved by the encoder and sent to the network. In fact, practice showed that sometimes the actual bit rate achieved varies significantly compared to the bit rate the Quality Adaptation module intended to achieve. Whether there is a big difference between the bit rate requested by the Quality Adaptation module and the actual bit rate, mainly depends on the encoder and the encoding format, as well as the capabilities of the server host.
  - <u>Packet scheduler/ Server buffer</u>: This module sends all outgoing information to the network. Its responsibility is to encapsulate data in RTP packets. There is also a buffer used to smooth accidental problems during the network transmission.
- **Client Manager**: Corresponds to a unique Client. It processes the RTCP reports generated by the Client and can be considered as a representative of the client at the side of the server. It can interact only with one Stream Manager at a given time, the Stream Manager controlling the stream from which the client is receiving the video. It contains the following component:
  - <u>Feedback analysis/Report buffer</u>: This module receives the RTCP reports from the client and processes them based on packet loss rate and delay jitter information. It then makes an estimation of the state of the client, based on the current and a few previous reports, that it stores in a buffer. The exact operation of the algorithm is described in a following paragraph.

2.2 Client Architecture

The client architecture consists of the following modules:

- **Client buffer**: Multimedia data received are first stored in this module, and presentation does not begin unless there is a necessary amount of data stored in the client buffer. In order to achieve smooth media presentation to the user, this buffer's capacity has to exceed the maximum delay jitter during data transmission.

- **Feedback**: This is the module that produces the information necessary for the Feedback Analysis Module at the server to estimate the client's state. Control information is transmitted with RTCP reports, which include, as mentioned earlier, information about packet loss rate and delay jitter.
- **Decoder**: This module takes the data packets from the client buffer as input, decodes them and outputs them suitable for presentation. The quality of the presented video is higher when the receiving data rate is high. Video quality can also be affected by packet loss and delay. Presentation can come to a complete stop if data in the client buffer drops below the required minimum
- **User Display**: The module responsible for the presentation of the video to the user, which can be a computer monitor.

3 Description of System Operation and Algorithms

The source initially constructs a number of streams. This number depends on the number of receivers with different reception capabilities that expected to request video from the source and the processing capabilities of the machine where the server runs. Since in this implementation this number is determined at the initialisation of the server, an estimation of the actual number of receivers is useful. A small pre-determined number can, however, work well in most cases, because it offers the possibility of a reasonable categorization of all receivers according to their capabilities.
Each stream has to do its own processing on the video so as to transmit it at its prescribed rate, therefore many streams mean heavy processing load for the server. A solution to this problem, apart from using a small number of streams, could be storing the video in various (or all) needed encoding types so that each stream can use its own locally stored file. This, of course, has the drawback of taking up more disk space. Since a small number of streams were considered satisfying for our experiments, our implementation performs encoding on the fly. A stream without any clients in its session is ready for transmission but remains inactive.
When a client wants to start receiving video, it requests from the server the address of a multicast session belonging to a transmitting stream. This and all other Client – Server communication is made through the use of CORBA (Common Object Request Broker Architecture). Since the client user can have some local knowledge over its connection quality, he can make an initial estimation of the stream in which the client might best fit and is allowed to enter any stream the user chooses. If no choice is made, then by default the client enters the lowest quality stream. An overestimation can be expected to have small negative effects, since the client will be moved shortly to a more suitable stream.
By joining a multicast session the client informs the stream to start transmitting, if it is not doing so already. A dedicated Client Manager is created to represent the client at the side of the server. RTCP reports are sent back to the stream and in particular to the appropriate Client Manager's Feedback Analysis module. Information in RTCP reports contains two values that describe the quality of the transmission: packet loss rate and delay jitter. These values are passed through the following filters used to avoid wrong estimations and determine the aggressiveness of the feedback analysis protocol:
For the packet loss rate:

$$LR_{new} = a * LR_{old} + (1-a) * LR_{net}$$

Where: LR_{new}: The new filtered value of packet loss rate, LR_{old}: The previous filtered value of packet loss rate. For the first report after the start of transmission, this value is 0, LR_{net}: The packet loss value that was contained in the RTCP report received from the Client. a: a parameter that determines the aggressiveness of the adaptation concerning the packet loss value. Its value ranges from 0 to 1, with a=0 meaning that only the current report is taken into account, and a = 1 meaning that all new RTCP reports are ignored.
For delay jitter:

$$J_{new} = b * J_{old} + (1-b) * J_{net}$$

Where: J_{new}: The new filtered value of delay jitter, J_{old}: The previous filtered value of delay jitter. For the first report after the start of transmission, this value is 0, J_{net}: The delay jitter that was contained in the RTCP report received from the Client. b: a parameter that determines the aggressiveness of the adaptation concerning the delay jitter value. Its value ranges from 0 to 1, with b=0 meaning that only the current report is taken into account, and b = 1 meaning that all new RTCP reports are ignored.

For the sake of clarity, a distinction has to be made between two kinds of states, that both can take the values of UNLOADED, LOADED or CONGESTED: we call the first one the "unprocessed state" and the second the "processed state". The unprocessed state is derived directly from the filtered values of packet loss rate and delay jitter, according to the following rules:

if $(LR_{new} >= LR_c)$ *unprocessed state = CONGESTED*
if $(LR_u < LR_{new} < LR_c)$ *unprocessed state = LOADED*
if $(LR_{new} <= LR_u)$ *unprocessed state = UNLOADED*
if $(J_{new} > \gamma * J_{old})$ *unprocessed state = CONGESTED*

We have defined LR_U as the maximum value of the unloaded packet loss rate and LR_C as the minimum value of the congested packet loss rate. Where γ is a parameter, which specifies how aggressive the network condition estimation component will be to the increase of delay jitter.

The state that will be reported to the Optimal Rate Estimation module of the Stream Manager is called the processed state. It is computed by taking into account the last *n* unprocessed states, which are held in an n-sized buffer in the Client Manager. This buffering mechanism contributes to the conservative behaviour of the Optimal Rate Estimation module. A CONGESTED unprocessed state does not necessarily impose that the processed state will also be congested, especially if the majority of the previous "unprocessed states" were UNLOADED. The way the processed state is computed is presented below:

We first introduce a new variable, USV (Unprocessed State Variable), that takes a new value for each unprocessed state as shown:

if $(unprocessed\ state_i == CONGESTED)$ *then* $USV_i = -1$
if $(unprocessed\ state_i == LOADED)$ *then* $USV_i = 0$
if $(unprocessed\ state_i == UNLOADED)$ *then* $USV_i = 1$

The processed state is then determined by the value of

$$f(i) = state_i * w_i + state_{i-1} * w_{i-1} + \dots + state_{i-n+2} * w_{i-n+2} + state_{i-n+1} * w_{i-n+1}$$

where $w_i, \dots, w_{i-n+1}$ are weights used to quantify the decreasing importance of old unprocessed states.

if $(f(i) < 0)$ *then processed* $state_i$ *= CONGESTED*
if $(f(i) == 0)$ *then processed* $state_i$ *= LOADED*
if $(f(i) > 0)$ *then processed* $state_i$ *= UNLOADED*

Information update in Client Managers is made asynchronously, every time an RTCP report arrives. However, Stream Managers update their rates synchronously and therefore time in system operation is divided in epochs of certain length. At the end of an epoch, each Stream Manager polls the states of all the Client Managers that correspond to a client receiving this stream and the Quality Adaptation module determines the improvement or degradation in this stream's video quality. Whether there will be an improvement or degradation is determined as follows: If all receivers[1] are in the UNLOADED state, video quality is improved. If more than a certain threshold of receivers is CONGESTED, video quality is degraded. The threshold used for our experiments was one-third of all receivers listening to the stream.

The new bit rate is estimated using an Additive Increase, Multiplicative Decrease (AIMD) algorithm, just like TCP. Increase is achieved by adding a standard small value to the previous bit rate, and is therefore quite conservative in bandwidth consumption, while decrease is achieved by multiplying the previous bit rate with a number in the range of 0…1 (typically around 0.75) and so the algorithm is more aggressive when trying to react to congestion.

[1] The number m of the receivers can easily computed by the RTCP protocol

There are three cases in this phase that will lead to a client's transition towards another stream:

- If the stream from which the client is currently receiving video has already reached its lowest transmitting rate and the client is still in CONGESTED state then the client stops listening to this stream and joins the session of a lower quality stream (if such a stream exists).
- If the stream from which the client is currently receiving video has already reached its highest transmitting rate and the client is still in UNLOADED state then the client stops listening to this stream and joins the session of a higher quality stream (if such a stream exists).
- The third case applies to a client that co-exists in a stream with low capacity receivers but is capable of handling better quality video, so it has been unable to improve the video quality of the current stream. The mechanism used aims in making the protocol more conservative and operates by counting the number of consecutive times the receiver was UNLOADED but failed to improve the video quality. When this number exceeds a certain limit, we assume that the receiver has indeed higher capabilities and move it to a better quality stream. Transition from one stream to another also means that the client's corresponding Client Manager module will now interact with the new Stream Manager.

The conservatism our protocol exhibits has two advantages: (1) We successfully ignore RTCP reports that are the result of temporary factors, for example congested reports that appear almost certainly when a stream transition occurs and are due to system load but have a very short effect on the reception capability. (2) Our protocol is TCP-friendly, because it only consumes excessive bandwidth when it is absolutely certain that this bandwidth can be handled, and furthermore uses the conservative AIMD algorithm.

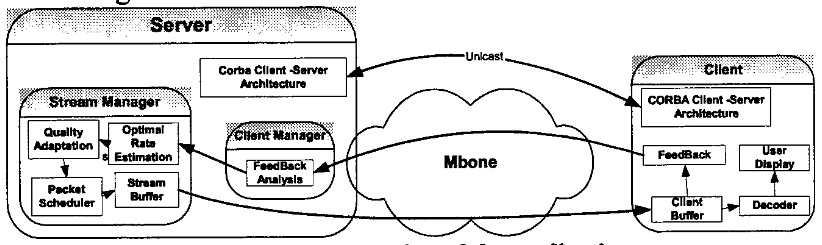


Figure 2 The operation of the application.

4 Implementation Issues

For the implementation of our system we used the Java Programming Language, and in particular the Java Media Framework API ([12]). Java's object-oriented model fits our design and JMF offers a convenient level of abstraction, which allows the developer to concentrate on high-level issues, thus making it an ideal platform for experimental research. In particular, JMF provides support for RTP transmission and reception of real-time media streams across the network. It offers some very useful classes and interfaces, like the Session Manager, that encapsulates the creation, maintenance and closing of an RTP session, the Processor that encapsulates processing and control of time-based media data and the DataSource that encapsulates media protocol-handlers. Our JMF-based implementation is represented by the Figure 3 and Figure 4. The Figure 5 shows the Graphical User Interface of the Client.

All communication between the server and the clients is achieved using CORBA. This technology allows a module written in any language that supports CORBA to be integrated seamlessly in our system, as long as it implements a small number of functions necessary for remote communication.

CORBA communication between the Server and the Clients also requires a third entity, the Naming Service. It can be located on the same host as the Server or on any other host in the network. All clients and the server, however, must know its location. When the Server is initialised, it registers itself and all the Stream Managers it creates to the Naming Service using a hierarchical representation similar to an operating system's file structure. When a client is started it uses the Naming Service to request a reference to the Stream Manager it wishes to receive data from. Every time the client makes a transition to a different stream, it uses the Naming Service to get a reference to the new Stream Manager. Since communication may also be directed from the Client to the Server, during initialisation every Client also registers itself to the Naming Service. This way the

Client Manager module (which is part of the Server entity) can locate its corresponding Client and order it to move to a different stream whenever necessary.
These choices generally indicate our purpose for this implementation to be experiment- and flexibility- oriented, rather than performance-oriented and therefore it can be improved in terms of resource optimisation.

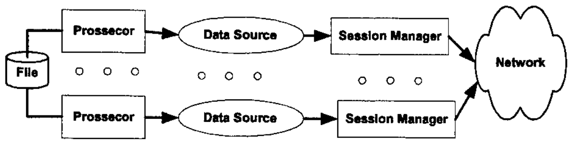


Figure 3 The Server operation

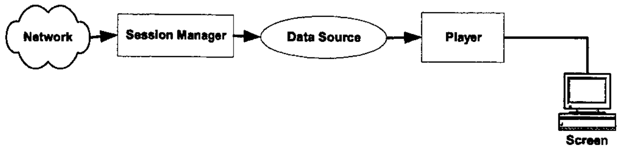


Figure 4 The Client operation

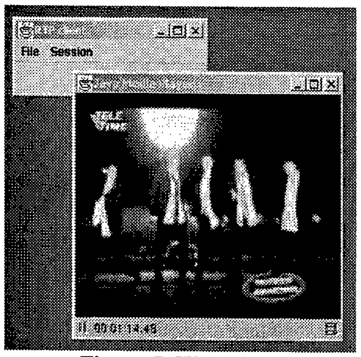


Figure 5 Client GUI

5 Performance Evaluation – Initial results

In order to evaluate the performance of the implemented prototype, we run an experiment over a controlled networking test-bed, which, we have implemented over the campus network of University of Patras in Greece. The test-bed consists of one Server and six Clients. We connect each participant with connections of different capacity to our network test-bed with the use of traffic policy on the access router of each participant (the Server and the six Client) in the test-bed. More particularly, the Server is connected to our test-bed with 2 Mbps connection and the Clients are connected with connections, which vary from 200-700 Kbps. The Server was transmitted three streams with the following limits: Stream 1: 10Kbps-184Kbps, Stream 2: 184Kbps-368Kbps and Stream 3: 368Kbps-600Kbps.During the experiment the Server was using the following parameters in order to control the operation of the implemented mechanism: a=0.5, b=0.8, γ=2, LR_u=0.02, LR_c=0.05 (the values of the above parameters base on experimental results). The AIMD algorithm of the Server was increasing the transmission rate of a Stream by 50Kbps during network unloaded periods and was decreasing the transmission rate to 75% during network congestion periods. We run the experiment for 360 sec and in the begging of the experiment the Server transmits only the Stream 1 with transmission rate of 10Kbps and all the Clients are connected to Stream 1.
Figure 6 shows the transmission rate of the Server Streams during the experiment and the Figure 7 shows the receive rate of a representable Client which is connected to the network test-bed with 400 Kbps connection.
As the Figure 6 shows, in the beginning of the experiment all the Clients are connected to Stream 1 and the transmission rate of Stream 1 increase until it reach its upper limit. Then some Clients switch to Stream 2 and the Server starts transmit the Stream 2 (at 60th second) except of the Stream 1. After some time some Clients switch to Stream 3 (at 100th second) and the Server transmits all the three streams. During the experiment, the Server stops a stream if the stream does not have any

receivers. As the Figure 7 shows, the representable Client starts receiving Stream 1 from the Server and when this Stream reach its maximum capacity, the Client switch to Stream 2 in order to exploit the available bandwidth. The switching from Stream 1 to Stream 2 takes some time to complete because the join action and especially the leave action of a multicast group take some time to complete.

The above described experiment can only be consider as an initial performance evaluation of the implemented prototype and shows predictable operation of the implemented prototype and has encouraging results. We plan to investigate in more detail the behaviour the implemented prototype in our future work.

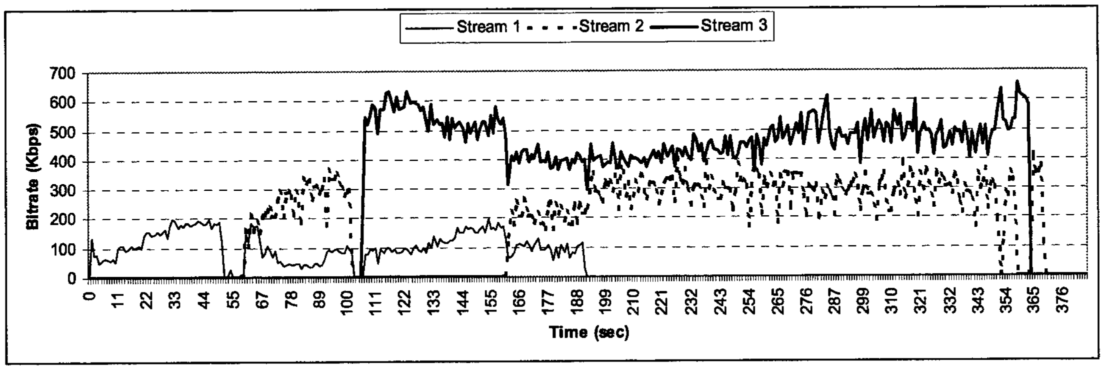


Figure 6 Transmission Rate of Server Streams

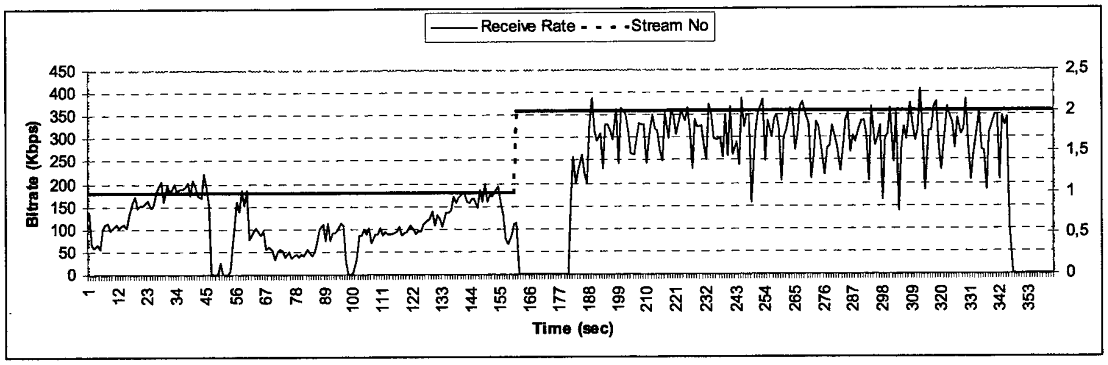


Figure 7 Receive Rate of representable Client

6 Conclusion - Future Work

In this paper, we present the implementation of a prototype for multicast transmission of adaptive multimedia data in a heterogeneous group of receivers with the use of replicated streams. We concentrate on the design of a mechanism for monitoring the network condition and estimate the appropriate rate for the transmission of the multimedia data in each stream in order to allocate each receiver to the appropriate streams and treat the receivers with fairness. The implemented prototype uses RTP/RTCP protocols for the transmission of multimedia data.

Our future work includes the validation of implemented prototype by using it for the multicast transmission of multimedia data in a heterogeneous group of receivers both in a controlled network test-bed and in the Internet in order to examine the behaviour of the implemented prototype against TCP and UDP traffic. In addition we plan to examine the behaviour of the proposed mechanism in very large multicast group through simulation. In addition we will investigate the benefits of dynamically adding more streams instead of the static number of streams that the implemented prototype supports now.

7 Bibliography

[1] C. Cowan, S. Cen, J. Walpole, C. Pu. "Adaptive Methods for Distributed Video Presentation", ACM Computing Surveys, 27(4), pp. 580-583, December 1995. Symposium on Multimedia.

[2] R. Rejaie, D. Estrin, and M. Handley, "Quality Adaptation for Congestion Controlled Video Playback over the Internet" in Proc. of ACM SIGCOMM '99, Cambridge, Sept. 1999.

[3] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, L. Yu, "A player for adaptive mpeg video streaming over the internet," in Proceedings of the 26th Applied Imagery Pattern Recognition Workshop AIPR-97, SPIE, (Washington DC), Oct. 1997.

[4] I. Busse, B. Deffner, H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP," Computer Communications, Jan. 1996.

[5] S. Jacobs, A. Eleftheriadis, "Adaptive Video Applications for Non-QoS Networks", Proc. 5 th International Workshop on Quality of Service (IWQoS'97), Columbia University, New York, USA, pp.161-165.

[6] R. Ramanujan, J. Newhouse, M. Kaddoura, A. Ahamad, E. Chartier, K. Thurber, "Adaptive Streaming of MPEG Video over IP Networks", Proceedings of the 22nd IEEE Conference on Computer Networks (LCN'97), November 1997.

[7] P. Mundur, A. Sood, R. Simon, "Network Delay Jitter and Client Buffer Requirements in Distributed Video-on-Demand Systems", Department of Computer Science George Mason University Fairfax, VA 22030.

[8] S. Cen, C. Pu, J. Walpole, "Flow and Congestion Control for Internet Media Streaming Applications", In Proceedings of Multimedia Computing and Networking, 1998.

[9] B. Vandalore, W. Feng, R. Jain, S. Fahmy, "A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia", Journal of Real Time Systems (Special issue on Adaptive Multimedia), April 99.

[10] Shculzrinne, Casner, Frederick, Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, IETF, January 1996.

[11] Shculzrinne, Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, IETF, January 1996.

[12] Java Media Framework: http://java.sun.com/products/java-media/jmf/index.html

[13] R. Rejaie, M. Handley, D. Estrin. "Architectural considerations for playback of quality adaptive video over the Internet", Technical Report 98-686, USC-CS, November 1998.

[14] R. Rejaie, M. Handley, D. Estrin. "RAP: An end-to-end rate-based congestion control mechanism for real time streams in the Internet", Proc. IEEE Infocom, March 1999.

[15] H. Smith, M. Mutka, D. Rover, "A Feedback based Rate Control Algorithm for Multicast Transmitted Video Conferencing", Accepted for publication in the Journal of High Speed Networks.

[16] J.-C. Bolot, T. Turletti, I. Wakeman. "Scalable feedback control for multicast video distribution in the Internet" In Proceedings of SIGCOMM 1994, pp. 139-146, London, England, August 1994. ACM SIGCOMM.

[17] T. Jiang, E. W. Zegura, M. Ammar, "Inter-receiver fair multicast communication over the Internet". In Proceedings of the 9th International Workshopon Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pp. 103-114, June 1999.

[18] B. J. Vickers, C. V. N. Albuquerque T. Suda, "Adaptive Multicast of Multi-Layered Video: Rate-Based and CreditBased Approaches," Proc. of IEEE Infocom, March 1998

[19] S. McCanne, V. Jacobson. Receiver-driven layered multicast. 1996 ACM Sigcomm Conference, pp. 117-130, August 1996.

[20] S. Floyd, K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," IEEE/ACM Transactions on Networking, 1998.

[21] T. Jiang, M. Ammar, E. W. Zegura, "Inter-Receiver Fairness: A Novel Performance Measure for Multicast ABR" Sessions. pp. 202-211 SIGMETRICS 1998

[22] X. Li, M. Ammar, S. Paul, "Video Multicast over the Internet", IEEE Network Magazine, April 1999

[23] S. Y. Cheung, M. Ammar, X. Li. "On the Use of Destination Set Grouping to Improve Fariness in Multicast Video Distribution", INFOCOM 96, March 1996, San Fransisco.

[24] Ch. Bouras, A. Gkamas, "Streaming Multimedia Data With Adaptive QoS Characteristics", Protocols for Multimedia Systems 2000, Cracow, Poland, October 22-25, 2000, pp 129-139.

[25] C. Diot - Sprint Labs "On QoS & Traffic Engineering and SLS-related Work by Sprint", Workshop on Internet Design for SLS Delivery, Tulip Inn Tropen, Amsterdam, The Netherlands, 25 - 26 January 2001.

[26] J. Pandhye, J. Kurose, D. Towsley, R. Koodli, "A model based TCP-friendly rate control protocol", Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Basking Ridge, NJ, June 1999.

[27] D. Sisalem, A. Wolisz, "LDA+ TCP-Friendly Adaptation: A Measurement and Comparison Study," in the 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'2000), June 25-28, 2000, Chapel Hill, NC, USA

Retrieval Scheduling for Multimedia Presentations* (Extended Abstract)

Martha L. Escobar-Molano†, David A. Barrett,
{mescobar,barrettd}@asgard.com
Asgard Systems

Zornitza Genova, Lei Zhang
{zgenova,lzhang}@csee.usf.edu
University of South Florida

Abstract

Advances in computer graphics, authoring tools and the explosive growth of the Internet has increased the use of multimedia presentations. This article presents a new retrieval scheduling technique to support the display of multimedia presentations in a multi-user environment. A multimedia presentation consists of a collection of objects with temporal constraints that define when the objects are rendered. A scheduling algorithm must determine when objects are retrieved from disk to satisfy the temporal constraints of the presentation. The time elapsed between the arrival of a request and the onset of its display (latency) depends upon the resources (CPU, disk, and memory) available to the system. The resources available depend upon those consumed by other presentations already being displayed. Therefore, the latency must be computed when the new request for a presentation arrives and that latency must include its computation time.

Prior scheduling techniques applicable to arbitrary resource requirements have quadratic time complexity. Unlike prior work, our scheduling algorithm has linear time complexity. We compare the performance of our scheduling technique with one that exhaustively searches for the earliest time to schedule a presentation. Our simulation results show that our technique significantly reduces the latency of a presentation as compared with the exhaustive search.

1 Introduction

A multimedia presentation consists of a collection of objects with temporal constraints that define when the objects are rendered. For example, a computer generated animation consists of a collection of 3-D objects that represent the characters and background of the animation with their time of appearances. To display a presentation, the storage system must deliver the participant objects to the renderer according to their time of appearances.

We assume that the unit of retrieval from disk into memory is a fixed-size page. Objects are not restricted by the page size. Smaller objects are clustered together into a page and larger objects are partitioned into multiple pages. We partition time into fixed-size time intervals. A retrieval schedule of a presentation defines for each time interval what pages to

*This research was supported, in part, by the National Science Foundation CAREER Award No. 9875528
†This work was done while this author was at the University of South Florida

retrieve from disk and what pages to discard from memory to satisfy the temporal constraints of the presentation. The retrieval schedule also determines the memory and disk bandwidth requirements of the presentation at each time interval.

The retrieval schedule overlaps the display of the presentation. To minimize memory requirements, the retrieval schedule aims to fetch pages from disk during the interval that preceeds their display. However, the disk bandwidth capacity might not be sufficient to retrieve all pages displayed at interval i during its preceeding interval $i - 1$. Therefore, some pages might be pre-fetched at an earlier time interval. Only pages displayed during the first time interval and pages required to be pre-fetched before the beginning of the presentation are retrieved before the display of the presentation.

The system maintains a *system availability* that contains the resources still available while presentations are being displayed. The system availability is a sequence of tuples, one for each interval that the system has allocated resources for presentations being displayed. For a system with D disks, each tuple has $1 + D$ elements and represents the available memory and bandwidth for each disk at the corresponding interval.

When a request for a presentation arrives, the system determines when to schedule the request so that the presentation requirements would not exceed the memory and disk bandwidth in the system availability.

An approach to determine when to schedule the request is to exhaustively search for the earliest sequence of time intervals when the presentation can be scheduled. Suppose that a request for a presentation with a retrieval schedule of k time intervals arrives at interval $t - \tau$, where τ is the time to search for this sequence. The system tries to start the retrieval schedule of the presentation at interval t. If there is a time interval when the memory and disk bandwidth requirements in the retrieval schedule exceed the the system availability, then it tries starting at t+1. If starting at t+1 also exceeds the availability, it tries at t+2, and so forth. When trying to start the retrieval schedule at t, the system compares pair-wise the system availability at intervals $t, t+1, \ldots, t+k-1$ with the requirements at intervals $0, 1, \ldots, k-1$ in the retrieval schedule. Since there are $D + 1$[1] resources, it takes $k \times (D + 1)$ comparisons to check whether the presentation can be scheduled starting at t. If the number of tuples in the system availability is n, it takes up to $k \times (D + 1) \times n$ comparisons to determine when the presentation requirements would not exceed the memory and disk bandwidth available in the system. Therefore the time complexity of the exhaustive search is quadratic.

To illustrate, consider a system configuration with 36 disks ($D = 36$) and a CPU of 400 MHz. Suppose that we partition time into 4-second time intervals. When trying to schedule a 2-hour ($k = 1800$ time intervals) retrieval schedule on a system that has allocated resources for 24 hours ($n = 21{,}600$ time intervals), the system performs up to $1,800 \times 37 \times 21,600 = 1,438,560,000$ comparisons. Suppose that it takes 23 cycles to compare the required and available amounts of a single resource for a single time interval. Then, it takes up to $1,438,560,000 \times 23/400,000,000 = 82.71$ seconds to search for the earliest sequence of time intervals when the presentation can be scheduled ($\tau \leq 82.71$). Since the system does not know in advance how long this search will take, it must consider the worst case ($\tau = 82.71$) to guarantee that all the temporal constraints of the presentation are satisfied. Starting the display before this search ends might introduce disruptions at the end of the presentation because the system might not have available resources to retrieve the pages on time for their

[1]One for each disk and one for memory.

display.

In this paper, we present a technique that determines when to schedule a request in linear time. Our technique limits the number of comparisons to $lf \times n \times (D+1)$, where lf is a linear factor. For a linear factor of 50, our technique takes up to $50 \times 21,600 \times 37 \times 23/400,000,000 = 2.30$ seconds to search for the time to schedule the presentation in the example above. As illustrated in this example, the search time following the exhaustive search is significantly higher than our technique.

2 Related Work

Previous studies [9, 1, 2] have investigated scheduling for continuous media (e.g., audio, stream-based video). These studies conceptualize a presentation as a file that is read sequentially at a pre-specified rate. They assume a data layout so that the disk reference pattern is regular, e.g., read the first block of a presentation from disks 0, 1, and 2 during the first time cycle, read the second block from disks 3, 4, and 5 during the second time cycle, and so on. Presentations sharing objects might reference them in different order. Therefore, finding a placement of objects that yields a regular disk reference pattern might be infeasible. Our scheduling technique works for arbitrary data layouts.

Retrieval scheduling for composite multimedia objects have been studied before [6, 8]. They conceptualize a presentation as a collection of multimedia streams with temporal constraints. However, their scheduling techniques have quadratic time complexity for presentations whose resource requirements (memory and disk bandwidth) vary arbitrarily over time.

Retrieval scheduling for presentations whose resource requirements vary arbitrarily over time have been studied before. In [5], an optimal retrieval scheduler for single disk architectures was proposed. This scheduler minimizes both latency and memory requirements. The complexity of resource scheduling for multi-disk architectures was studied in [4]. This study demonstrated that the computation of a resource schedule that satisfies a pre-specified display schedule and minimizes the startup latency is NP-Hard. In [3], a taxonomy of resource scheduling techniques that satisfy the display schedule of a presentation was proposed. This study introduced three resource scheduling techniques for multi-disk architectures and quantified their trade-offs. However, both [5] and [3] assume a single-user environment.

In a multi-user environment, the system has to find a sequence of time intervals where the memory and disk bandwidth available is sufficient to support the display. This problem can be stated as follows: given a sequence of requirements $r_1, \ldots, r_{m'}$, find a subsequence in the system availability $sys_{i+1}, \ldots, sys_{i+m'}$ such that for each $j \in [1, m']$: $r_j \leq sys_{i+j}$. For multiple resources, sys_j and r_j are vectors. We can pose the problem of finding a match for a string in a document in a similar manner. The string corresponds to the requirements and the document to the system availability. And, the matching criteria is equality: for each $j \in [1, m']$: $r_j = sys_{i+j}$. Like the string matching technique in [7], our scheduling techniques encodes the scheduling decision process into an Finite State Automaton (FSA). However, having $\leq$ as the matching criteria makes the encoding and execution of an FSA significantly different from the encoding and execution in the string matching problem.

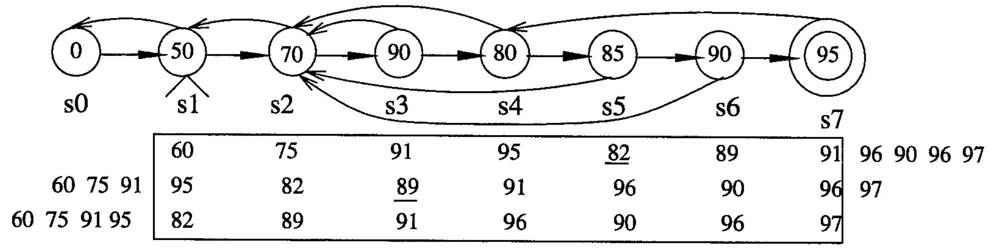


Figure 1: An example of an FSA for a single resource with a system availability being recognized by the FSA.

3 Scheduling Techniques

The proposed scheduling technique pre-computes the memory and disk bandwidth requirements of a presentation. The single-user retrieval scheduling techniques in [3] can be used to precompute these requirements. Based on these requirements, it builds an FSA and stores it on disk. This FSA recognizes the sequence of intervals when the resources (disk bandwidth and memory) in the system availability are greater than or equal to the resources required by the presentation. When a request arrives, the system loads and executes the FSA to determine when the presentation can be scheduled.

An FSA is a tuple $(P, \delta_f, \delta_b, s_1, s_{m'})$, where P is the set of states, δ_f and δ_b transition mappings, s_1 the starting state, and $s_{m'}$ the accepting state. If the retrieval schedule of a presentation requires m' time intervals, its FSA has $m' + 1$ states $(s_0, s_1, \ldots, s_{m'})$. There is a state for each time interval in the retrieval schedule and a sentinel state (s_0). Each state (s_i) is associated with the memory and bandwidth requirements of the presentation at the corresponding time interval (i). Transition mappings δ_f and δ_b represent forward and backward transitions, respectively. Forward transitions occur when the memory and disk bandwidth at the current time interval in the system availability are greater than or equal to the requirements associated with the current state in the FSA. Backward transitions occur where the requirements associated with the current state in the FSA exceed the system availability at the current time interval. The FSA starts at state s_1 and reaches state $s_{m'}$ when recognizes the sequence of intervals in the system availability when the presentation can be scheduled.

An FSA can be represented as a directed graph. To illustrate, suppose that we have only one resource in our system (say memory). Figure 1 represents the FSA for a presentation whose memory requirements for each interval are 50, 70, 90, 80, 85, 90, 95. Each state is represented by a node (circle) with its requirements inside. The state with a zero in it is the sentinel state. Forward and backward transitions are represented by forward and backward links, respectively. The starting state is marked by a carat (s_1) and the accepting state is represented by a double circle (s_7).

Because of space limitations, we describe the algorithms here and refer the reader to the full paper for the pseudo-code of the algorithms.

3.1 Running the FSA

When a request for a presentation arrives, the system loads the FSA associated with the presentation and executes it. The FSA reads the first tuple in the system availability (memory and bandwidth available during the first time interval) and compares it with the requirements in state s_1 (memory and bandwidth required by the presentation during the first time interval). If the tuple in the system availability is greater than or equal to[2] the requirements in s_1 (the current state), the FSA advances to the next state by following a forward link and reads the next tuple from the system availability. If the tuple in the system availability is smaller than the requirements in the current state, the FSA advances to the next state by following a backward link but does not read the next tuple from the system availability. It continues comparing tuples in the system availability with the requirements in the states of the FSA until it reaches the accepting state or the end of the system availability. Following a backward link from s_i to s_j increases the latency by $i - j$.

To illustrate consider the example in Figure 1. The system availability is represented by the sequences of numbers below the FSA. When the FSA starts, it reads 60 from the system availability and compares it with 50 (the amount in s_1). Since $60 \geq 50$, the FSA follows the forward link and reads the next element in the system availability (75). Similarly, it compares $75, 91, 95$ with $70, 90, 80$, advances to state s_5, and reads 82. Since $82 < 85$, the FSA follows the backward link from s_5 to s_2. Following this backward link increases the latency by three time intervals. This increase is represented by a shift of 3 time intervals in the system availability as shown in the second line at the bottom of Figure 1. Then, it compares 82 with 70. Since $82 \geq 70$, the FSA follows the forward link and reads the next element in the system availability (89). Since $89 < 90$, the FSA follows the backward link from s_3 to s_2. Following this backward link increases the latency by one time interval, as shown in the third line at the bottom of Figure 1. Then $89, 91, 96, 90, 96, 97$ are compared with $70, 90, 80, 85, 90, 95$ and the FSA reaches the accepting state (s_7). Therefore, the request can be scheduled during the sequence of time intervals with $82, 89, 91, 96, 90, 96, 97$ as their memory available. The total latency is four time intervals. The sentinel state (s_0) is used to force reading the next element in the system availability. For example, suppose that the FSA runs on a system availability whose first element is 45. Since $45 < 50$, the FSA follows the backward link to s_0 and compares 45 with 0. Since $45 \geq 0$, it reads the next element in the system availability.

As shown in the full paper, the complexity of the algorithm to execute an FSA is $\mathcal{O}(n)$, where n is the number of tuples in the system availability.

3.2 Constructing the FSA

When constructing the FSA, a forward link is created between the states associated with two consecutive time intervals in the retrieval schedule of the presentation. When executing an FSA, the tuples in the system availability are compared with the requirements of the presentation in sequential order. Once a tuple t satisfies the requirements of the request in the current state, a new tuple is read and t is not referenced by the FSA anymore. For

[2]A tuple in the system availability is greater than or equal to the requirements in a state if and only if: (1) the memory available in the tuple is greater than or equal to the memory required in the state, and (2) for each disk d, the bandwidth available for d in the tuple is greater than or equal to the bandwidth required for d in the state.

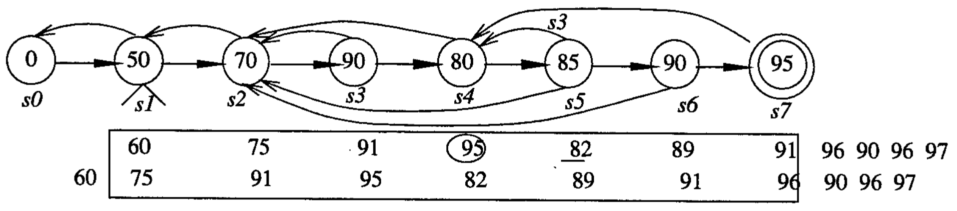


Figure 2: An example of an FSA for a single resource

example when processing 82 in the system availability, the FSA followed the backward link from s_5 to s_2 and did not have to verify that the memory available in the preceding time interval of the system availability (95) is greater than or equal to the requirements in s_1 (50). Therefore, the FSA must be constructed so that following a backward link during execution does not require comparing preceding tuples in the system availability with corresponding requirements. Thus, the FSA is constructed so that there is a backward link from s_i to s_j having the shortest distance $i - j$ that satisfies the following two conditions:

(i) The requirements in s_i are greater than the requirements in s_j, and

(ii) For all states s_k such that $1 \leq k < j$, the requirements in s_k are smaller than or equal to the requirements in s_{k+i-j}.

3.3 Refinements

To reduce the latency incurred by a presentation, we introduce additional backward links (termed *conditional links*). A conditional link from state s_i to s_j is represented by a labeled link. The labels specify the condition that must be satisfied in order to follow the link. A label is a sequence of states $s_{i_1}, \ldots, s_{i_k}$ whose requirements might be higher than the system availability. In order to follow a conditional link, the system must verify that the requirements in states $s_{i_1}, \ldots, s_{i_k}$ are smaller than or equal to the system availability. Conditional links do not satisfy Condition (ii) in Section 3.2. The labels in the link are the states s_k that violate Condition (ii). Figure 2 shows a conditional link from s_5 to s_4 with label s_3. A backward link from s_5 to s_4 increases the latency by one time interval. Therefore, the requirements of each pair of consecutive states before s_5 are compared to obtain the states that violate Condition (ii). Since s_3 is the only state that violates Condition (ii) ($s_3 > s_4$), the label of this link is s_3. This label will be used during execution to determine whether or not to follow the link. To illustrate, suppose that we are executing the FSA in Figure 2. During this execution, it reads 82 (in the system availability) and compares it with the requirements in s_5 (85). Since $82 < 85$, then it follows a backward link. There are two backward links: from s_5 to s_4 and from s_5 to s_2. It first tries the shortest link (from s_5 to s_4). Since this link is conditional, it checks first whether there is enough memory available to satisfy the requirements of s_3. If the link is followed, the element in the system availability that would correspond to s_3 is 95. Since $95 \geq 90$, the link is followed.

An FSA with conditional links is a tuple $(P, \delta_f, \delta'_b, s_1, s_{m'})$. δ'_b maps a state s to a sequence $L = [< s_{k_1}, seq_{k_1} >, < s_{k_2}, seq_{k_2} >, \ldots, < s_{k_l}, seq_{k_l} >]$ of pairs. Each pair $< s_{k_i}, seq_{k_i} >]$ represents a target state and the label on the link. The sequence L is sorted by the increase of latency incurred by the link: $k_1 > k_2 > \ldots > k_l$. A backward link from s_i to s_{k_1} increases the latency by $i - k_1$, while a link to s_{k_2} increases the latency by $i - k_2$. Hence, the higher the subscript $(k_1, \ldots, k_l)$ is the lower the latency is.

The algorithm to execute an FSA with conditional links selects the backward link with the shortest latency such that the condition on its label is satisfied. Checking this condition increases the number of comparisons performed by the FSA. Therefore, our algorithm to construct the FSA selects the backward links that would bound the total number of comparisons during execution of the FSA to a linear factor lf of n $(lf \times n)$, where n is the number of tuples in the system availability. This bounding is achieved by selecting the backward links such that the number of comparisons while transiting a cycle from state s_{i-j} to s_i back to s_{i-j} is less than or equal to $lf \times j$. The full paper describes this bounding in detail.

There might be more than one set of backward links that satisfy the above selection criteria. To decide which link to include in $\delta'_b(s_i)$, our technique assigns priorities to each link. The priority of a link from s_i to s_j with label L is defined as $1 - \frac{|L|}{j-1}$. The value of $|L|$ is bounded by $j - 1$, therefore $\frac{|L|}{j-1}$ represents the percentage of number of states in the label over all possible states. The lower this percentage is the higher the priority of the link.

4 Evaluation

We compared our scheduling technique with a scheduler that exhaustively searches for the earliest time intervals in the system availability when the presentation can be scheduled. The performance of both techniques was evaluated using a simulation study and synthetic data.

As described in the full paper, we generated display schedules for 16 presentations of 100 minutes, 16 of 45 minutes, and 16 of 40 seconds. Once the display schedules of the 48 presentations were generated, we applied the memory-based scheduling technique in [3] to compute the memory and disk bandwidth requirements of each presentation assuming four different system configurations: (1) 1 GBytes of memory and 12 disks, (2) 2 GBytes of memory and 24 disks, (3) 3 GBytes of memory and 36 disks, and (4) 4 GBytes of memory and 48 disks. All configurations have a single CPU of 400 MHz and a page size of 128 KBytes. Each disk supports a 338.1 mbps transfer rate, 11.24 millisecond seek time, and 6 millisecond rotational latency.

The computed memory and disk bandwidth requirements are used by the exhaustive search to find the earliest time to start the retrieval schedule. These requirements are also used to build an FSA for each presentation. The FSAs were constructed with linear factors of 100 for 100-min and 45-min presentations and 6 for 40-sec presentations.

We assumed that the frequency of access of the presentations follows the Zipf distribution and the inter-arrival time of requests follows the Poisson distribution. Based on these assumptions, we generated lists of requests for arrival rates varying from 0.1 to 4.0 arrivals per minute. The span of request arrivals for these lists was 2 hours. We also assumed that besides displaying presentations on demand, the storage system supports pre-scheduled presentations. For our simulations, we assumed that the system pre-scheduled presentations for 8 hours.

We then schedule the requests in each list using both approaches: the exhaustive search and the FSA-Based technique. For each request, we compute the latency incurred by the presentation. This latency has 5 components:

(1) Time to retrieve pages referenced at the beginning of the display and to pre-fetch some pages

Table 1: Computation Time for Exhaustive Search vs FSA.

Movie Length	12 Disks ES	12 Disks FSA	24 Disks ES	24 Disks FSA	36 Disks ES	36 Disks FSA	48 Disks ES	48 Disks FSA	Linear Factor
40	0.064	0.032	0.124	0.064	0.184	0.092	0.244	0.120	6
2700	3.644	0.540	7.008	1.036	10.376	1.532	13.740	2.028	100
6000	8.088	0.540	15.552	1.036	23.016	1.532	30.484	2.028	100
Avg	4.422	0.433	8.587	0.822	12.620	1.213	16.645	1.606	NA

Computation time for the FSA algorithm depends upon a implementor-chosen linear factor. Avg is the average over the mix of all arrivals in the simulations. Values other than the Linear Factor are in seconds.

in preparation for the display. This component is derived from the memory-based scheduler and is identical for both approaches.

(2) Time to compute when to start the retrieval schedule of current request. This computation is dominated by the comparisons between available resources in the system and the requirements of the presentation. Therefore, we assume that this component is the time taken by the comparisons. For the exhaustive search, this time is $k \times n \times (D+1) \times 23/CPUfreq$, where k is the number of time intervals in the retrieval schedule of the presentation, n is the number of tuples in the system availability, D is the number of disks, 23 is the number of cycles per comparison, and $CPUfreq$ is the number of cycles per second. For the FSA-Based, this time is $lf \times n \times (D+1) \times 23/CPUfreq$, where lf is the linear factor.

(3) Delay due to a CPU conflict with the scheduling of previous requests. When a new request arrives while the CPU is still searching for the time to schedule a previous request, the system has to wait until the search is finished before it starts the search for the new request.

(4) Time to retrieve meta-data from disk. For the exhaustive search, this component is the time to retrieve the file containing the memory and bandwidth requirements of the presentation. For the FSA-Based, this component is the time to retrieve the file containing the FSA computed by our technique.

(5) Delay due to memory or disk bandwidth conflicts with other presentations being displayed. This component is the increase of latency due to shortage of resources in the system availability for the presentation requirements. For the FSA-Based, this component is the latency incurred when following the backward links.

The latency incurred by the FSA-Based technique does not include the time to build the FSA. The construction of the FSAs is done before the requests of presentations arrive. This construction is based on the memory and bandwidth requirements of each presentation, which is pre-computed (Section 3).

4.1 Results

Table 1 shows how the average time to compute the retrieval schedule (component 2) for our FSA algorithm compares with the exhaustive-search algorithm. In all cases the FSA consumes less computation time and the gap widens significantly as the as the movie length grows.

We seek to determine whether this reduction in computation for FSA will result in lower latency than exhaustive-search. Because FSA limits the number of comparisons by a linear

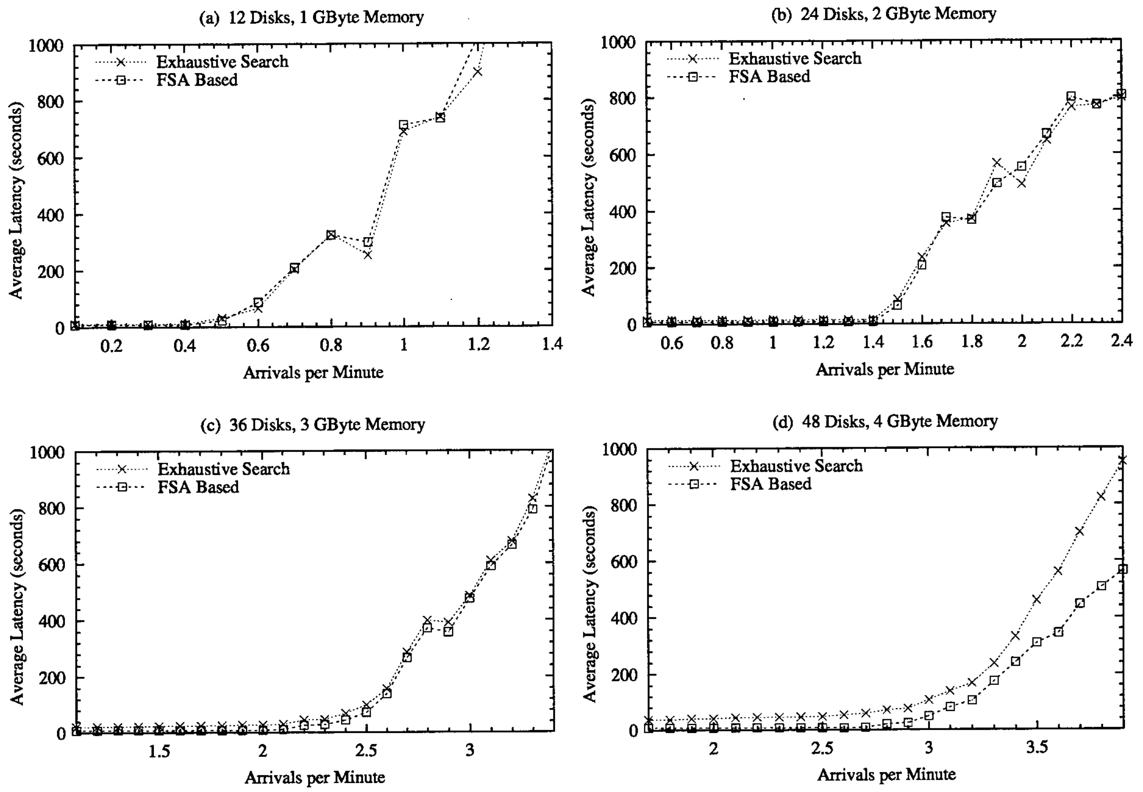


Figure 3: Total Latency. Total net average latency incurred for presentations as a function of request arrival rate for four system configurations.

factor, it will occasionally incur increased latencies because of resource conflicts with disk bandwidth or memory requirements.

Figure 3 shows the total latency for all components (1-5 combined). In Figure 3(a), for arrival rates below 0.5, the average total latency for our FSA technique was 8 seconds and varied from 10.7 to 11.2 seconds with exhaustive-search. Since the exhaustive-search always finds the earliest time intervals to schedule a presentation, the delay due to resource conflicts with other presentations is expected be lower for the exhaustive-search. For arrival rates greater than .5, the average latency with exhaustive search was up to 24% lower than with our scheduling technique (occurred at .6 arrivals per minute, too small to see in the figure).

In Figure 3(b), for arrival rates smaller than 1.5, FSA incurred 8 seconds total latency and exhaustive-search varied from 14.2 to 15.5 seconds. For all arrival rates, the average latency with exhaustive search was up to 13% lower (@1.9Arr/min) than the latency with our scheduling technique. In Figure 3(c) FSA slightly outperformed exhaustive-search for all arrival rates. CPU computation time (Table 1) and CPU contention (component 3) start to have a stronger influence for this configuration. Figure 3(d) shows that our FSA technique significantly outperformed the exhaustive-search algorithm for all arrival rates. The average latency of exhaustive-search was between 39% and six times higher (@2.7 Arr/min) than FSA. The average latency varied from 38 to 951 seconds for exhaustive-search and from 8.5 to 569 seconds for FSA. Computation time was significant for this configuration. Table 1 shows average computation time for exhaustive search was 16.6s as compared to 1.6s for FSA. Arrival rates of higher than 3.6 requests per minute yield inter-arrival times of less than 16.6

seconds. Therefore exhaustive-search suffers progressively further delays because the CPU is busy computing previous schedules when new requests arrive.

Thus, our FSA algorithm's improvement in computation time translates into a significant reduction in latency over exhaustive-search in this case.

5 Conclusions

This paper introduced a linear retrieval scheduling technique to support the display of multimedia presentations. We compared the performance of our technique with a scheduling technique that exhaustively searches for the earliest time intervals where the request can be scheduled. Simulation results show that the reduction on the computation time of our technique results in lower latencies (up to six times lower) than the exhaustive search, as the number of resources (disks) in the system increases.

Our scheduling technique reduces the computation time by budgeting the number of comparisons during the scheduling of a presentation. This paper introduced one alternative to budget the comparisons. One question that arises is how different alternatives to budget comparisons affect the outcome of the scheduling technique. Other future research directions include: retrieval scheduling techniques that maximize throughput, retrieval scheduling for dynamically generated display schedules such as in video games, and fault tolerant retrieval scheduling techniques.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM-SIGMOD*, pages 79–89, May 1994.

[2] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi. Avoiding retrieval contention for composite multimedia objects. In *Proceedings of Very Large Databases*, 1995.

[3] M. L. Escobar-Molano and S. Ghandeharizadeh. On Coordinated Display of Structured Video. *IEEE Multimedia*, 4(3):62–75, July-September 1997.

[4] M. L. Escobar-Molano and S. Ghandeharizadeh. On the Complexity of Coordinated Display of Multimedia Objects. *Theoretical Computer Science*, 242(1-2):169–197, 2000.

[5] M. L. Escobar-Molano, S. Ghandeharizadeh, and D. Ierardi. An Optimal Resource Scheduler for Continuous Display of Structured Video Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):508–511, June 1996.

[6] M. N. Garofalakis, Y. E. Ioannidis, and B. Ozden. Resource Scheduling for Composite Multimedia Objects. In *Proceedings of Very Large Databases*, pages 74–85, August 1998.

[7] D. E. Knutt, J. H. Morris, and V. A. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[8] C. Shahabi, S. Ghandeharizadeh, and S. Chaudhuri. On Scheduling Atomic and Composite Multimedia Objects. *IEEE Transactions on Knowledge and Data Engineering*, To Appear.

[9] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, pages 393–400, August 1993.

Distributed Multimedia Information Retrieval that Accepts Arbitrary Media Key

Takashi Hayashi(takasi@dq.isl.ntt.co.jp)*, Gengo Suzuki(gsuzuki@dq.isl.ntt.co.jp)*, Yuichi Iizuka(iizuka@rd.nttdata.co.jp) **, Kazuya Konishi(koni@dq.isl.ntt.co.jp) * and Takashi Honishi(honishi@dq.isl.ntt.co.jp) *

In this paper, we propose an integrated retrieval system that retrieves distributed multimedia information from existing heterogeneous information sources by entering any media key independent of the sources. The main feature of the proposed system is a media translation dictionary that can translate the medium of the retrieval key entered into media appropriate for the available information sources by connecting different media data. Furthermore, it issues query statements compatible with each heterogeneous information source to retrieve distributed multimedia information. In addition, the dictionary can be automatically made from learning data. We implement a prototype system and perform image and text integration retrieval experiments. The effectiveness of making the dictionary automatically is confirmed.

1. Introduction

The amount of multimedia information continues to increase because documents, images and sounds are rapidly being converted into digital format. Contents are stored in files and distributed databases as information sources. Retrieving the information needed by entering any multimedia key would yield a very useful retrieval method. For example, you could retrieve the title of a movie by humming the theme song, or you could retrieve a report by entering a keyword or drawing a figure.

In this paper, we propose an integrated retrieval system that allows distributed multimedia information to be retrieved from existing heterogeneous information sources by entering any multimedia key.

* NTT Cyber Space Laboratories (1-1Hikari-no-oka, Yokosuka-Shi, Kanagawa, 239-0847 Japan)
** NTT DATA CORPORATION (1-21-2 Shinkawa, Chuo-ku, Tokyo, 104-0033 Japan)

2. Current Retrieval Methods and their Problems

As the amount of multimedia information continues to grow, concern is mounting over the development of easy and convenient retrieval methods. Most retrieval methods utilize text data to reference multimedia information. Data is retrieved by entering keywords as the retrieval keys. On the other hand, several studies have examined similarity retrieval [1][2][3]. In this method, characteristics are extracted directly from data and stored in a database. For example, when a user inputs an image key, the system calculates image similarity values or distances and retrieves similar images. Similarity is calculated by mapping from an N-dimensional characteristics space to a one-dimensional distance space. This mapping involves weight vectors because we must indicate which characteristics are to be emphasized. It is presumed that data that are close to each other are similar (Fig.1, hypothesis 1). Therefore, goal of similarity retrieval is to find data that are within a certain distance from the retrieval key [4]. Needless to say, however, it is the user who ultimately judges whether the retrieval results are similar to the retrieval key or not.

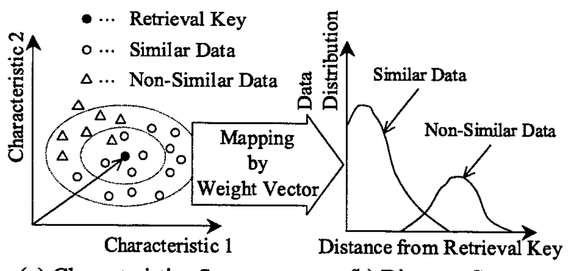


Fig.1 Similarity Retrieval in Characteristics Space and Distance Space

The problem that these retrieval methods have in common occurs when a user wants to retrieve objects using a retrieval key that is not suitable for some media types of information sources. In order to solve this problem, several studies have tried to connect data in different media types. In [5], linear mapping between text (adjectives) vector and image characteristics vector is calculated using canonical correlation analysis. This mapping has been used to connect texts and images. In [6], text data are allocated in a word space organized by a clustering method and the distance between texts is embedded in an image space. Sentences related to the input image are retrieved via similar images that are linked to texts. In [7], textual and visual characteristics are combined in a single index vector.

In order to retrieve distributed multimedia information from existing heterogeneous information sources by entering any multimedia key, there are two problems that must be overcome.

(1) It is difficult to connect data that have a multimodal distribution in a characteristics space

Current methods assume that the data that are similar to each other are close in the characteristics space (the inverse of hypothesis 1). However, this hypothesis does not hold if the data that should be connected have a multimodal distribution that makes it difficult to find a linear weight vector that can make them close. For example, let us consider how to connect red and green apple images to

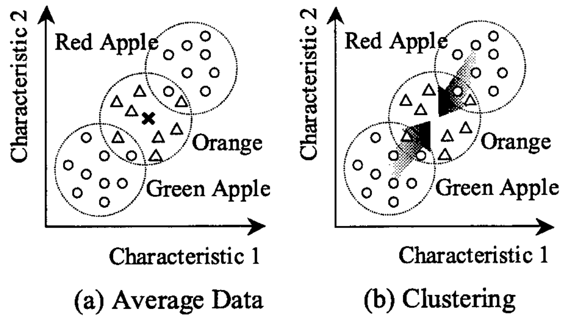


Fig.2 Connecting data in Current Methods

texts about apples. In this case, average data, which is calculated by averaging the characteristic vectors of red and green apple images, or clustering, are adopted in current methods (Fig.2). This makes it difficult to connect data that have a multimodal distribution in characteristics space.

(2) It is difficult to utilize existing heterogeneous information sources

The characteristics and query languages used differ from information source to information source, especially multimedia databases. A user must make query statements compatible with each type of information source. This makes it difficult to retrieve multimedia information from distributed heterogeneous information sources.

3. Proposed Method

To solve the problems mentioned in chapter 2, we propose an integrated retrieval system that allows any multimedia key to be used to retrieve multimedia information from distributed heterogeneous information sources. First, to translate the medium of the retrieval key into an appropriate medium for each information source, we show how to connect data of different media types. Second, we will discuss how to make query statements compatible with each heterogeneous information source so as to retrieve distributed multimedia information. Finally, we will discuss how to make connections automatically from learning data consisting of complex media types.

3.1. Media Translation using Dictionary

It is the user who is the final judge of whether data of different media types resemble each other or not. As mentioned in chapter2, however, similar data are not always close in the characteristics space. Even if similar data have a multimodal distribution in the characteristics space, it is necessary to connect similar data and perform media translation. To utilize various information sources, the data itself is better than the characteristic vectors because the kind of characteristic vector differs from information source to information source. Therefore, we propose a media translation method that uses a dictionary that connects similar data in different media type through an identifier with the same category (Fig.3). We call the data of different media type 'dictionary key' and the identifier of a category an 'entry.' An 'entry' can be explained by 'dictionary keys' in media translation dictionary just like a regular dictionary; furthermore, the same dictionary key connects different entries according to the user's interpretation.

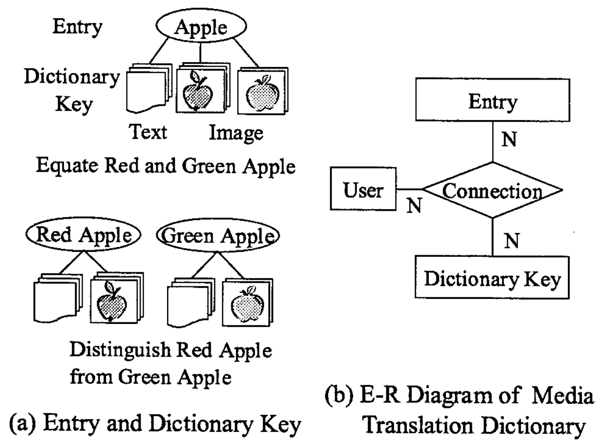

Entry
Apple
Dictionary Key
Text
Image
Equate Red and Green Apple
Red Apple
Green Apple
Distinguish Red Apple from Green Apple
(a) Entry and Dictionary Key
Entry
N
User
Connection
N
N
Dictionary Key
(b) E-R Diagram of Media Translation Dictionary

Fig.3 Media Translation Dictionary

Therefore, it is necessary to express many-to-many relationships between entries and dictionary keys in the media translation dictionary as set by users (Fig.3 (b)).

In addition, it is easy to accommodate new types of media, because the uniform data structure can be used to manage connections between data. In short, only the dictionary keys for the new type of media have to be added to an entry to accommodate a new type of media.

3.2. Steps in Retrieval

Fig.4 shows the retrieval steps using media translation.

3.2.1 Media Translation Function

A media translation dictionary consists of entries, dictionary keys, and connections as set by users. The media translation function is realized using three sub-functions:

(1) Query interpretation function;
(2) Entry look up function;
(3) Key look up function.

We describe these functions in detail below.

(1) Query interpretation function

A retrieval key and the dictionary keys are seldom exactly the same because it is impossible to manage all data as dictionary keys. Therefore, it is necessary to retrieve the dictionary keys that are similar to the retrieval key with the same type of media using similarity retrieval.

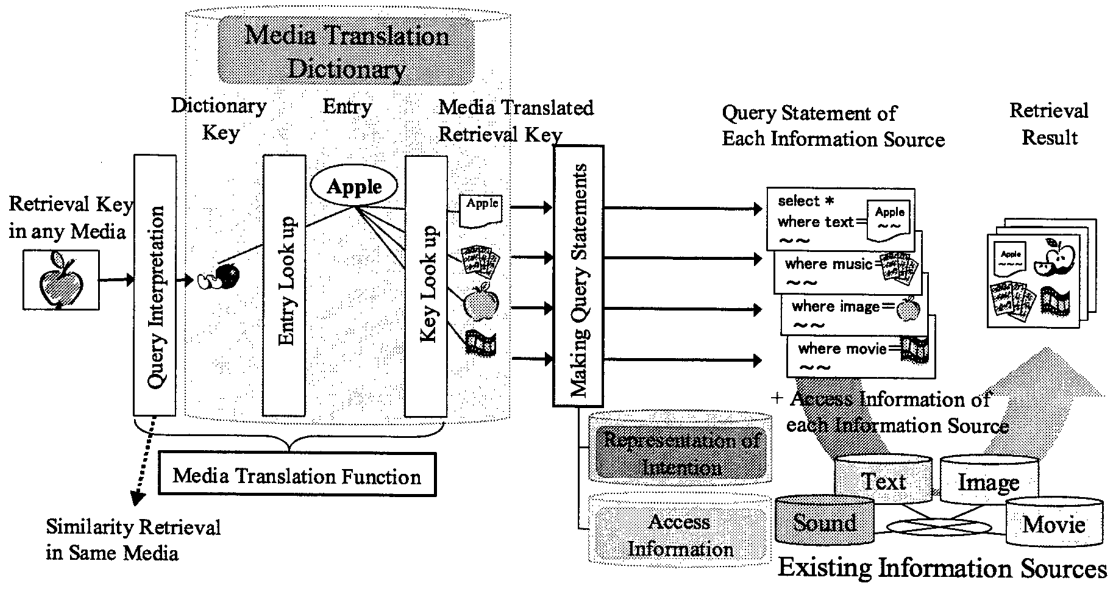

Media Translation Dictionary
Dictionary Key
Entry
Media Translated Retrieval Key
Query Statement of Each Information Source
Retrieval Result
Retrieval Key in any Media
Query Interpretation
Entry Look up
Apple
Key Look up
Making Query Statements
select *
where text=
where music=
where image=
where movie=
+ Access Information of each Information Source
Representation of Intention
Access Information
Text
Image
Sound
Movie
Existing Information Sources
Media Translation Function
Similarity Retrieval in Same Media

Fig.4 Steps in Retrieval

We call this function, the query interpretation function.

(2) Entry look up function

To determine retrieval keys' meaning, entries are looked up by the dictionary keys. We call this function, the entry look up function.

(3) Key look up function

To translate the medium of the retrieval key into a different media, the dictionary keys of different media types are looked up by entries. We call this function, the key look up function.

3.2.2 Query Statement Function

The proposed system must make query statements that are compatible with heterogeneous distributed information sources using the media-translated dictionary keys. However, the characteristics and weight vectors differ from information source to information source, especially multimedia databases. Furthermore, most query languages are extensions of SQL. Fig.5 shows query statement examples of ImageCompass [1] and QBIC [2], which are image similarity retrieval engines. It appears necessary to meet two requirements:

(1) Making query statements that support the user

If a user emphasizes different characteristics, the retrieval results will differ. For example, there are many cases in which the user emphasizes color or shape in image similarity retrieval. To get good retrieval results, it is necessary to make the query statements follow the user's intention. Of course, each query statement must suit the type of information source accessed.

(2) Accommodating new information sources

Access information, such as logical address, user name and so on, differs from information source to information source. If a new information source is added as a retrieval target, the proposed system should dissolve heterogeneity and accommodate the source.

To meet the first requirement, the proposed system manages weight vectors

Query Statement Example of ImageCompass

```
select x.filename
 from Image x, x.subimages y
 where y.features similar(40,'0hue',0.8,'L1',•••)
   SubImageFeatures.MakeValue('0hue,17,0.1,0.2,•••');
```

Query Statement Example of QBIC

```
select ObScoreFromStr
 ('ObColorFeatureClass file=<server, "filename">'
   weight=50.0 and
    ObTextureFeatureClass weight=30.0, db2image),
     Content(db2image, 'jpg'), FILENAME(db2image)
   from db2image_table order by 1;
```

Fig.5 Query Statement Examples of ImageCompass and QBIC

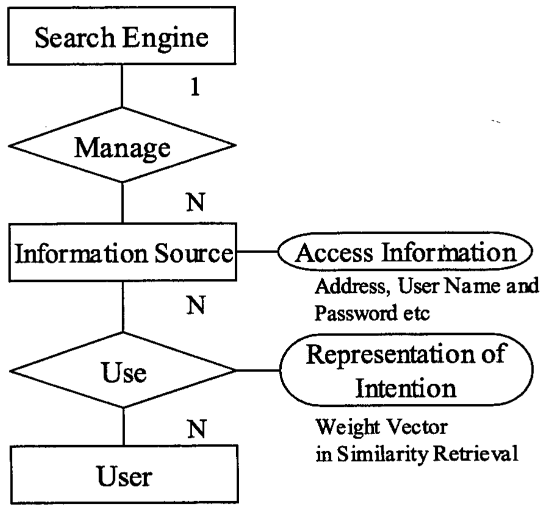
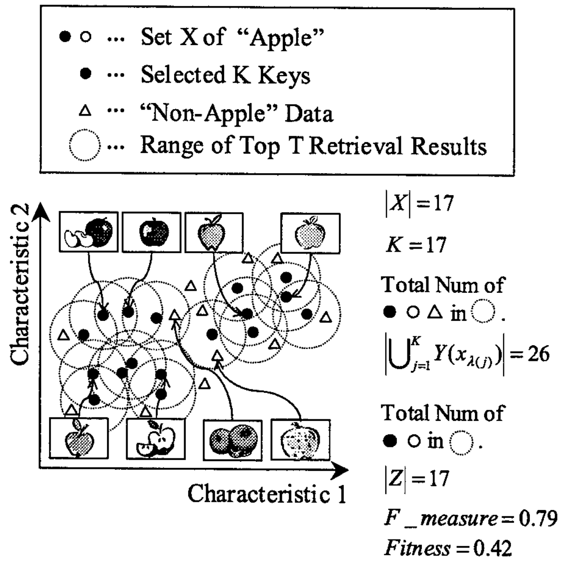

Search Engine
1
Manage
N
Information Source
Access Information
Address, User Name and Password etc
N
Use
Representation of Intention
N
Weight Vector in Similarity Retrieval
User

Fig.6 Representation of Intention and Access Information

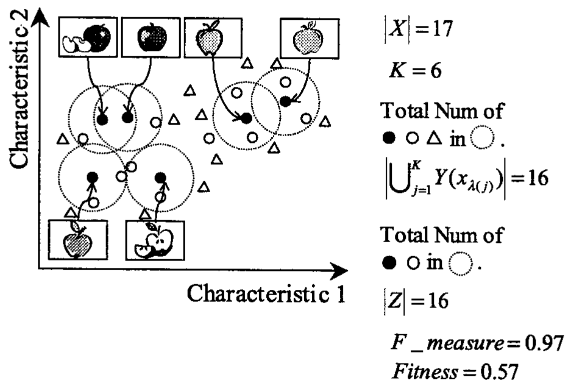
as ‘representation of intention.’ To meet the second requirement, it manages logical address, user name and so on of each source as access information (Fig.6). Query statements compatible with each type of information source are made from the media-translated dictionary keys, referring to the weight vector. Referring to the access information, query statements access each information source.

3.3. Automatic Dictionary Construction

The amount of work needed to make a media translation dictionary manually is huge. We propose a method that makes the dictionary automatically from learning data in complex media types, for example, web pages, electronic dictionaries and so on. Effective dictionary keys can be defined as 'data that represents an entry.' For example, in [5], average data in the characteristics space represents an entry. However, this method is not effective against data that have multimodal distributions. Our approach is to choose effective dictionary keys from learning data by estimating whether the retrieval results are good or not. Estimation of retrieval results is based on F measure, which is the weighted average of recall ratio and precision ratio [8]. In addition, the number of effective dictionary keys should be minimized. This approach is a form of combinatorial optimization problem. Genetic algorithms [9], simulated annealing [10] and so on have been proposed as ways to solve the



(a) Case where All Keys are Selected



(b) Case where Effective Keys are Selected

Fig.7 Selection of Keys

combinatorial optimization problem. An automatic dictionary construction method can be created by combining one of these solutions with an appropriate fitness function. Our idea is to maximize the F measure and minimize selected key number as follows.

A set X is a set of dictionary keys that are connected to an entry, and each dictionary key is element x_i. It is the user who judges whether this connection is valid or not. Therefore, a set X is a learning data, and relevance set as well. $Y(x_i)$ is the set of top T retrieval results using each x_i as the retrieval key. At first, we assume $K(\neq 0)$ keys are selected, we denote these keys as $x_{\lambda(1)}, x_{\lambda(2)}, \cdots, x_{\lambda(K)} (\lambda(j) \in \{1,2,\cdots,N\})$. The union of retrieval results using selected K keys is calculated;

$$\bigcup_{j=1}^{K} Y(x_{\lambda(j)})$$

Intersection Z of a set X and $\bigcup_{j=1}^{K} Y(x_{\lambda(j)})$ is calculated;

$$Z = X \cap (\bigcup_{j=1}^{K} Y(x_{\lambda(j)}))$$

Precision, Recall, and F measure are calculated;

$$Precision = \frac{|Z|}{\left|\bigcup_{j=1}^{K} Y(x_{\lambda(j)})\right|}$$

$$Recall = \frac{|Z|}{|X|}$$

$$F\_measure = \frac{1}{\dfrac{\alpha}{Precision} + \dfrac{1-\alpha}{Recall}}$$

$|A|$ expresses the number of elements in set A; $\alpha(0 \leq \alpha \leq 1)$ expresses whether emphasis is placed on precision or on recall.

We define minimization of selected key number as follows;

$$1 - \frac{K}{|X|}$$

Our proposed fitness function that maximizes F measure and minimizes selected key number is as follows;

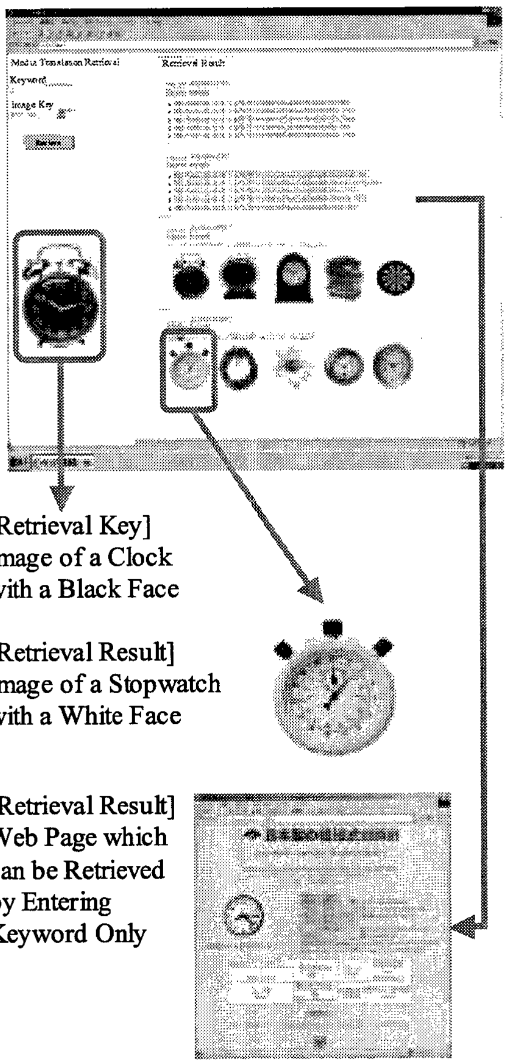


Fig.8 Example of Retrieval using Prototype System

$$Fitness = \beta \times F\_measure + (1-\beta)\left(1 - \frac{K}{|X|}\right)$$

$\beta(0 \leq \beta \leq 1)$ expresses whether emphasis is placed on maximization of F measure or on minimization of selected key number. When the fitness function becomes more than a set value, $x_{\lambda(1)}, x_{\lambda(2)}, \cdots, x_{\lambda(K)}$ are selected as effective dictionary keys. Repeating this process, for all entries, all users, and all

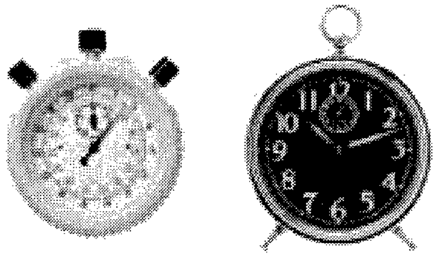


Fig.9 Selected "Clock" Keys

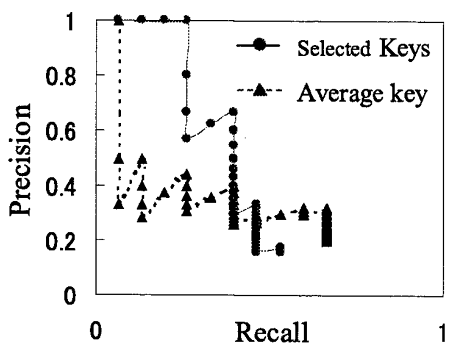


Fig.10 Recall-Precision Graph of Selected Keys and Average Key

media types, can automatically make a media translation dictionary. Fig.7 shows the selection of effective keys using image data connected to the entry "apple." Fig.7(a) shows the selection of all keys; fig.7(b) shows the selection of the most effective keys.

4. Experiments

4.1. Implementation and Retrieval Experiments

We implemented a prototype system. This system can retrieve image and text information if either an image key or a keyword is entered. The image information source, PhotoDisk [11] images, is managed by ImageCompass [1]; the text information source is managed by namazu [12]. These search engines are also utilized by the "Query interpretation function." In addition, web pages, which can be retrieved by entering keywords only, can be used as another information source. All we have to do is add the representation of intention and access information.

Fig.8 shows retrieval examples of the prototype system. Entered image key is a clock with a black face. Dictionary keys, image and text, are looked up by entering the image key into the media translation function. Similar image retrieval and full text search are done automatically by entering these dictionary keys. Retrieval results of ImageCompass are a clock with a black face and a black darts target etc. On the other hand, clocks with white faces are retrieved through the media translation dictionary. In addition, web pages can be retrieved via the image key. The result shows that integrated retrieval of image and text information is possible. The retrieval experiments confirmed that the media translation function was realized using dictionary keys of different media types. In addition, the proposed query function can realize integrated retrieval of information sources managed by different search engines.

4.2. Evaluation of Automatic Dictionary Construction

Experiments were conducted to evaluate the effectiveness of the automatic dictionary construction method. 640 images, including human faces, animals, instruments, and so on, were placed into an image database managed by ImageCompass [1]. Let us assume 15 clock images are connected to a "clock" entry. Effective 'dictionary keys' were selected. Fitness function, mentioned in 3.3, was optimized using a genetic algorithm. We determined α and β through a pilot experiment. Fig.9 shows 2 effective dictionary keys that offer the maximum fitness function (=0.43). Fig.10 shows a recall-precision graph of selected keys and average key, which was calculated by averaging the characteristic vectors of the 15 clock images.

Generally speaking, it is presumed that the data that are close to each other in the characteristics space are similar as determined by current retrieval methods. However, similar data are not always close in the characteristics space. Fig.10 shows that the retrieval results achieved with optimized keys are better than the ones achieved with the average key. Therefore, effective retrieval against multimodal distribution in the characteristics space was realized using multiple dictionary keys.

5. Conclusion

In this paper, we proposed an integrated retrieval system that retrieves distributed multimedia information from existing heterogeneous information sources by entering any multimedia key. We implemented a prototype system. This system can retrieve image and text information by entering either an image key or a keyword.

Our proposed method can be applied to other forms of multimedia as they are. When the proposed system integrate a music retrieval system [13] and full-text search engine, both hummed tunes and keyword of music can be accepted as queries, for example.

Since we use a media translation dictionary to connect different media data, the proposed system can translate the medium of the retrieval key into the media types appropriate for existing information sources. Since it makes appropriate query statements for each information source, it can retrieve distributed multimedia information from heterogeneous information sources.

In addition, the media translation dictionary can be automatically constructed from learning data by optimizing the fitness function. Evaluation experiments showed that effective retrieval against multimodal distributions in characteristics spaces was realized using the optimized dictionary keys.

References

[1] K. Kushima, M. Satoh, H. Akama and M. Yamamuro, Integrating Hierarchical Classification and Content-based Image Retrieval –ImageCompass-, In Proc. of Conference on Intelligent Information Processing, pp179-187, 2000.

[2] IBM Almaden Research Center: "Query by Image and Video Content: The QBIC System", IEEE Computer, Vol.28, No.9, pp.23-32, Sept 1995.

[3] M. T. Maybury, Editor, "Intelligent

Multimedia Information Retrieval", The MIT Press, 1997.

[4] Christos Faloutsos, "Searching Multimedia Databases by Content", Kluwer Academic Publishers, 1998.

[5] T. Kurita, T. Kato, I. Fukuda and A. Sakakura, "Sense Retrieval on an Image Database of Full Color Paintings", Information Processing Society of Japan, Vol.33, No11, pp.1373-1383, November 1992.

[6] Y. Mori, H. Takahashi, Y. Nitta and R. Oka, "Proposal of a Method for Image Understanding using Self-organized Data of Image and Text", Technical Report of IEICE, PRMU98-74, pp.9-15, September 1998.

[7] Marco La Cascia, Saratendu Sthi and Stan Sclaroff, "Combining Textual and Visual Cues for Content-based Image Retrieval on the World Wide Web", IEEE Workshop on Content-based Access of Image and Video Libraries, June 1998.

[8] van Rijsbergen, C. J., "Information Retrieval(2nd Ed.)", Butterworths, 1979.

[9] D. E. Goldberg, "GENETIC ALGORITHMS", Addison-Wesley Publishing Company, 1989.

[10] R.H.J.M. Otten and L.P.P.P. van Ginneken, "The annealing algorithm", Kluwer Academic Publishers, 1989.

[11]http://www.photodisc.com,
PhotoDics, Inc.

[12] http://www.namazu.org/index.html

[13] Naoko Kosugi, Yuichi Nishihara, Tetsuo Sakata, Masashi Yamamuro and Kazuhiko Kushima, "A Practical Query-By-Humming System for a Large Music Database", Proceedings of the 8th ACM International Conference on Multimedia, pp333-342, October 2000.

Towards a Flexible Information Retrieval Approach based on the Context

Francesca Arcelli Fontana
University of Milano-Bicocca,
email: arcelli@disco.unimib.it

Ferrante Formato
University of Salerno
email:formato@unisa.it

Abstract

In this paper we briefly describe a language, we have developed ([2]), for flexible information retrieval to deductive databases. It is a fuzzy logic programming language, where similarity is introduced on predicate and constant names and both flexible queries and crisp queries with flexible answers are allowed. Then we introduce our ongoing research directed to involve the notion of context in the query-answering process. We start from the observation that similarity is based on the context and we describe how an adaptive and flexible user interface based on the context can be developed. Finally, we outline the relevance of similarity-based search according to the context in many applications domains, as for web search and multimedia database systems.

Keywords: fuzzy techniques, similarity, flexible information retrieval, adaptive user interface.

1 Introduction

It is a largely recognized and obvious problem that with all the large amount of available information through the web, it becomes even more necessary to find some flexible information retrieval techniques, which allow us to find the more relevant and valuable information, to gather also the data that partially satisfy our requests and to be able to manage imprecise queries, since often user queries represent only an approximation of what one wants to know. To do that, both the notions of similarity and of context, as we will see below, become crucial for these information retrieval aims. Find the most relevant documents and information depends on the context of the query, while usually the result of a search is obtained independently of the context in which the request is made.

The role of the "context", and the knowledge derived from it, is a major topic of investigation at present in several areas, as knowledge-based systems, database systems, information retrieval and for different applications domains (see for example [9]). As the role of "similarity" for information retrieval and similarity-based search has been largely studied in the literature too.

In this paper we describe a flexible information retrieval approach to deductive databases, where flexibility is introduced in the data and in the queries through similarity, defined as a fuzzy equivalence relation. A fuzzy logic programming language, called Likelog (LIKeness in LOGic) has been defined in [2] for this purpose, where the core of the language is represented by the similarity-based unification algorithm. Logic programming is well suited for the representation of nested relations and to model complex types and objects. Through the fuzzy logic programming paradigm which we have proposed we are able to combine logic programming and databases allowing the management of uncertain or fuzzy information.

We observe, through some examples, how the answer to a query is strictly connected to the context in which this query is done and so we investigate in the paper how the role of the context can be captured through similarity too. Contexts may determine the true or falsity of a sentence as well as its meaning. We use similarity for two different but correlated aims: similarity notions are necessary for user query specification and satisfaction and similarity is particularly useful to model contextual knowledge.

We describe our ongoing research on the development of a system with a flexible and adaptive user interface, able to give flexible answers to the queries and to determine the context that best fits the user queries. Our approach to model the contextual knowledge is based on similarity and we give some hints on how to use the similarity reconstruction for adaptive user models able to guessing what the user wants. The availability of an efficient adaptive similarity-based search and user interface assumes a relevant role also for multimedia and distributed database systems.

As we said before, many works have been proposed in the literature on similarity-based research, in particular for multimedia databases, where similarity is captured through different approaches; for example in [10] similarity indexing has been described to scale large video databases and in [7] a fuzzy query language for multimedia data has been described, by introducing a similarity algebra, which extends relational algebra, incorporating the use of weights in predicates and operators to better fit user requirements. Another interesting approach is described in [1], where a multi similarity algebra is introduced for similarity based retrieval in multimedia databases. What kind of similarity measures could be used for multimedia databases, "ranking portions of the database with respect to similarity with the query" is deeply explored in [8].

The paper is organized through the following Sections: in Section 2 we briefly introduce the principal features of our similarity-based logic programming language Likelog and we show how flexible information retrieval based on similarity is allowed through this language; in Section 3 we describe the relations between similarity and context and we introduce how the similarity query/answering process could be extended according to the context. Finally we conclude and discuss several future developments related in particular to multimedia databases and on how the notion of context could be exploited to enhance web search.

2 Similarity-based Information Retrieval

We now describe how we can exploit similarity for flexible information retrieval to deductive databases through Likelog. Deductive databases can be viewed as information systems where knowledge is stored both explicitly, through facts, and implicitly, through a set of rules. Data are extracted through "query evaluation" methods, such as resolution and bottom-up evaluation. The similarity-based resolution used by Likelog is a flexible evaluation method for answering queries that cannot be satisfied in the classical case.

We now briefly describe the theoretical background of Likelog; for a deeper description of the language see [2].

A *t-norm* is a commutative and associative binary operation $*$ on the interval $[0,1]$ such that for any $x, y \in [0,1]$, $x * y \leq x' * y$ if $x \leq x'$ and $x * 1 = x$.

Definition 1 *Given a t-norm* $*$ *and a set* $\mathcal{U}$, *a* $*$*-similarity on* $\mathcal{U}$, *or simply a similarity, is a fuzzy relation, i.e. a map* $\mathcal{R} : \mathcal{U} \times \mathcal{U} \longrightarrow [0,1]$ *such that, for any* $x, y, z \in \mathcal{U}$ *:*

$\mathcal{R}(x,x) = 1$ *(reflexivity),* $\mathcal{R}(x,y) = \mathcal{R}(y,x)$ *(symmetry) and* $\mathcal{R}(x,z) \geq \mathcal{R}(x,y) * \mathcal{R}(y,z)$ *($*$-transitivity).*

We use the minimum as t-norm and in the following we will use the notations $\wedge$ and $\vee$ to indicate the *infimum* (*inf*) and *supremum* (sup), respectively.

We call *clouds* the non-empty subsets of $\mathcal{U}$. Intuitively, a cloud is a set of elements in $\mathcal{U}$ that are considered pairwise "similar". For example, consider the set $S = \{$*computer_science, network_economy, information_science*$\}$, where *computer_science, network_economy, information_ science* correspond to some of the several degrees available at the University level.

We set $\mathcal{R}$(*computer_science, network_economy*) $= 0.5$, $\mathcal{R}$(*computer_ science, information_science*) $= 0.8$, $\mathcal{R}$(*information_science, network _ economy*) $= 0.6$. Therefore the cloud $\{$*computer_science, information_science*$\}$ indicates a "set of University degrees" that are considered similar one another.

The degree up to which a cloud collapses into a singleton is called *codiameter;* given a cloud X, the codiameter $\mu(X)$ is defined as follows:$\mu(X) = \bigwedge_{x,y \in X} \mathcal{R}(x,y)$. One can easily extend μ to the empty set by setting $\mu(\emptyset) = 1$.We use the term "co-diameter" since the function $\mathcal{R}'(x,y) = 1$- $\mathcal{R}(x,y)$ is associated to a cloud, whose related diameter is given by 1- $\mu(X) = \bigwedge_{x,y \in X} \mathcal{R}'(x,y)$.

Let $\mathcal{L}$ be a first-order language with a set of variables $\mathcal{V}$, a set of constants $\mathcal{C}$, a set of predicate symbols $\mathcal{P}$ and no function symbols. We denote with $\mathcal{L}'$ the language obtained by replacing the set $\mathcal{C}$ in $\mathcal{L}$ with the set $\mathcal{C}'$ of non empty clouds of $\mathcal{C}$. Let $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{P}$ be a similarity on $\mathcal{C}$ and $\mathcal{P}$, respectively, $\mathcal{I}_\mathcal{V}$ be the characteristic function of the identity in $\mathcal{V}$ and let $\mathcal{R} = \mathcal{R}_\mathcal{C} \cup \mathcal{I}_\mathcal{V} \cup \mathcal{R}_\mathcal{P}$.

Usually, a complete definition of $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_\mathcal{C}$ is a task too large to be tackled by the user. For example, for any $x \in \mathcal{U}$, one could implicitly assume that $\mathcal{R}(x,x) = 1$. Analogously, if $\mathcal{R}(x,y)$ has been already defined, then $\mathcal{R}(x,y) = \mathcal{R}(y,x)$. Therefore, it can be convenient to let the user introduce $\mathcal{R}$ as a general fuzzy relation on a relatively small number of pairs of elements and let the system compute the smallest similarity $\overline{\mathcal{R}}$ that contains $\mathcal{R}$. We call *extended term* (briefly e-term) any element of $\mathcal{V} \cup \mathcal{C}'$ and then we derived a notation for our extended unification theory, defining what we mean for *extended substitution (e-substitution),* for *unification degree* and for *extended most general e-unifier (e-mgu).* The classical unification algorithm has been extended towards an unification under conditions (the clouds defined above) and the classical resolution rule has been extended using the similarity based unification algorithm, starting from a straightforward extension of the classical SLD-resolution for definite programs.

We have defined the operational semantics of Likelog given by the extended SLD-resolution and the fixed point semantics of Likelog given by the extension of the least Herbrand model of a program; it has been proved that the extension of the least Herbrand model corresponds to the fuzzy subset of ground formulae that are derived by extended refutations, obtaining in this way the completeness result of the coincidence of the operational and fix-point semantics of Likelog.

2.1 Flexible Information Retrieval

We now show, how it is possible to exploit the features of Likelog for flexible information retrieval to deductive databases. Consider, for example, the following instance of a deductive database for personal recruitment, where we have people classified according to their University degree and age and where we assume that on the market the most requested persons are those very young with computer_science degree.

```
computer_science(spot).
computer_science(strogoff).
computer_science(smith).
network_economy(bond).
network_economy(pearl).
information_science(boston).
law(giordano).
political_science(bianchi).
very_young(boston).
age(boston, 22).
age(bond, 28).
age(smith, 29).
age(strogoff, 32).
age(pearl, 34).
age(rossi, 31).
age(giordano, 40).
very_requested(X) :- computer_science(X), very_young(X).
young(X) :- age(X, N), N <= 30.
```

Moreover, we can consider other rules, as for example in relation to the University where the degree has been taken, by associating a value of importance to the University and by substituting the rule of `very_requested(X)` with another one in which also the "University-value" is taken into account.

In a classical database, if the user asks for a "very requested" person, no answer will be provided. The goal ?-`very_requested(X)`has no classical solution, since there is no person with the computer-science degree which is also `very_young`. Nevertheless, it seems reasonable for the user to consider the constants `very_young` and `young` as "similar" up to a certain degree, as also consider the similarity that exists between the different degrees.

Precisely, suppose that the user introduces the following similarities $\mathcal{R}$:

-$\mathcal{R}$(`computer_science`, `network_economy`) = 0.7

-$\mathcal{R}$(`computer_science`, `information_science`) = 0.8

-$\mathcal{R}$(`network_economy`, `information_science`) = 0.5

-$\mathcal{R}$(`law`, `political_science`) = 0.8

-$\mathcal{R}$(`law`, `computer_science`) = 0.1

-$\mathcal{R}$(`very_young`, `young`) = 0.9

-$\mathcal{R}(a,b) = i_A(a,b)$ where $A =$ {`spot`, `strogoff`,`smith`,`bond`, `pearl`, `boston`, `rossi`,`giordano`, `bianchi`}. and i_A is the classical identity on A.

Now, the query ?-`very_requested(X)`can be evaluated through Likelog, giving as a result the following pairs of computed extended substitution answers and conditions sets (obtained through the similarity-based unification algorithm):

<{X→ `boston`}, {`computer_science`, `information_science`} >,

<{X→`smith`},{`ery_young`, `young`} >,

<{X→ `bond`}, {`very_young`, `young`},

{`computer_science`, `network _ economy`} > .

This means that the user accepts the person "`boston`" as very requested provided that `computer_science` is similar to `information _ science`; in other words, the similarity degree between `computer _ science` and

`information_science` (which we call co-diameter of the cloud {`computer _ science`, `information _ science`}) represents the cost one must pay to have a classical refutation of `very_requested(X)`with X→ `boston` as computed answer substitution. Of course, to avoid possible infinite loops, the user must set a threshold as degree of validity of the answers of his query. The above query has been evaluated with degree 0.8 and a threshold can be supplied directly with the query. Moreover, if we ask for several people with information-science degree and we find only a person with this degree or suppose that we don't find any one, then we try to satisfy the request by finding all the people with the degree more similar to `information_science`, always applying our similarity-based research.

We have described a very simple example, in which obviously the conditions to consider a person as "very requested" and the similarity relations involved are subjective and can be changed, as one wants, but the example aims to outline the capabilities of Likelog for flexible data retrieval which can be exploited in many different and significant applications. We have studied applications in the case of an Estate Agency and of a Library.

3 Context-based Information Retrieval

There is no general agreement on what is to be considered a "context", most often, this notion is somewhat vague. Respect to our aims we are looking for a model of contextual knowledge that, starting from some sample features of a certain environment, (i.e. some samples of evaluation of distances or similarities) is able to derive a more general set of features allowing to "predict" the behavior of a certain "user". A context must be able to predict some actions and must be scalable; both these two features seem to be easily grasped by the notion of similarity. We now see how our definition of context can be modeled through similarity, exploiting the duality existing between the notions of distance and similarity.

To model a context with a similarity $\mathcal{R}$, suppose to have a set S, representing the whole context, and a finite set $N = \{P_1, P_2, ..., P_n\}$ of elements of S. Consider the case where the similarity $\mathcal{R}$ is described by "examples" i.e. the values $\mathcal{R}(x, y)$ are defined for any $x, y \in N$. Our goal is to determine the t-norm $*$ that best fits the behavior of $\mathcal{R}$ as a $*$-similarity or find the t-norm $*$ such that $\mathcal{R}$ is the $*$ -similarity that best fits the modeling of the context. Until now we have seen how similarity can be used for information retrieval aims exploiting in particular the similarity-based unification algorithm and the notions of cloud and co-diameter of cloud, which represent the extent by which a set does not contains dissimilar elements. Then, we observed how the notion of context is strictly connected to similarity. Now we introduce our aims in exploiting similarity to define a similarity-based information retrieval system according to the context. We start by view how we can model a context by using different t-norms.

In [3] we have pointed out two ways by which the background knowledge or experience of a user can be used, in order to discover the context. Either we give a comprehensive set of t-norms, and then we try to pick out the one that best fits a given fuzzy relation, or we give a parametric t-norm and then we try to fit the fuzzy relation through the choice of a suitable parameter.

In this case, one can choose between three different t-norms, the t-norm of the minimum, the t-norm of the product and the t-norm of Lucasiewicz, defined as $x * y = \max(x + y - 1, 0)$, which model the three most significant cases of conjunction in fuzzy logic ([5]). Three significant cases of contextual knowledge can be modeled using these t-norms.

For example, consider the case where we have to model the role of the context associated to a scenario, in which a tourist visiting a city wants to know where the different monuments are located respect to his position or respect to downtown, asking for these distances at a Tourist Board Office. After obtaining this information, the tourist will retain worthy to move or not.

When we have to consider "small" variations respect to the total distance considered, we may use the t-norm of the minimum, for large-scale distance, where the degree of closeness get lower and lower, we can use the t-norm of product and finally in the case in which beyond a certain distance, everything is considered "far", Lucasiewicz logic seems suitable to model this context. In this way we have observed how a restricted family of t-norms can models a wide variety of contexts. In turn, such t-norms can be parametrized to constitute a bundle of contexts, according to the specifications of context given before. This suggests a method to fine tuning of a context through a family of t-norms indexed by a parameter, since searching for the best parameter is easier than searching for the appropriate t-norm.

We would like to use this kind of similarity reconstruction to develop a similarity-based user model for flexible query/answering system. The similarity relations existing between the data in the database are established in advance by the system administrator or can be supplied directly by the user, through a fuzzy relation that the system extends into a similarity automatically. We are working on the possibility to develop a *similarity-based adaptive user interface* by which the system can directly learn from the user the similarities existing between the data, without the direct intervention of the user itself. In this case the user is able to choose directly a t-norm or the system is able to determine the t-norm that best fits the context described by the user through samples values. The system asks the user about the degree of similarity between objects of the domain and then according to the user answers, the system deduce the contextual knowledge and the t-norm which best fits it.

4 Conclusions and Future Developments

In this paper we described our ongoing research directed to study how similarity can be exploited for information retrieval purposes in two particular directions:

i) to allow a flexible query answering process able to retrieve the most satisfying answer, also when the classical approach fails; ii) to exploit the contextual knowledge to retrieve the more appropriate answer according to the context. In particular we focused our interest in exploring the development of a user adaptive interface able to guess the context of the user queries. Some works have been proposed for automatically infer search context; for example in [4] a completely different approach is described which attempts to model the context of user needs according to the content of the documents searched.

We are interested to explore how our research can be applied for web search. Many current web search engines are similar to traditional information retrieval systems according to the principal operations performed. In searching on the web, usually the results of the queries are independent from the "kind" of user, in particular from the context in which the user acts. We would like to explore how the contextual knowledge can be used to increase the web search, by developing a *similarity based web search engine* able to specifically request the information necessary to reconstruct a context and find the best answers. An interesting exploration on the role of the context for web search is described in [6].

Through the web a lot of multimedia data is available, we would like hence to experiment our approach to retrieve multimedia data through flexible queries, when crisp notions or exact matching are not satisfied or allowed and a notion of similarity is necessary to model the contextual domain and to model the imprecision and incompleteness in the representation of multimedia data.

In general, we think that database queries are typical situations where contextual knowledge occurs. We invite to observe that the role of the officer at the Tourist Board described in Section 3 can be satisfactorily played by an information system that exploits contextual information and this is not featured by classical databases and information retrieval systems.

Acknowledgments

The research reported is partially supported by the "MUSIQUE: Infrastructure for QoS in Web Multimedia Services with Heterogeneous Access", MURST project.

References

[1] S.Adali, P.Bonatti, M.L.Sapino and V.S.Subrahmanian. A Mullti-Similarity Algebra. *Proceedings of 1998 ACM SIGMOD Int.Conference on Management Data*, Seattle, WA, 1998, pp.402-413.

[2] F.Arcelli and F.Formato. "A Logic Programming Language for Flexible Data Retrieval". *Proceedings of ACM SAC'99*, San Antonio, Texas, 1999.

[3] F.Arcelli and and F.Formato. "User Adaptive Models based on Similarity", *Proceedings of ACM Symposium.on Applied Computing*, Como, Italy, 2000.

[4] J.Budzik and K.J.Hammond. User interfaces with everyday applications as context for just in time information access. *Proceedings of the 2000 Int. Conference on Intelligent User Interfaces*, ACM Press, New Orleans, Louisiana, 2000.

[5] P.Hajek. *Mathematics of Fuzzy Logic.* Kluwer Academic Publishers 1997.

[6] S.Lawrence. Context in Web Search. *IEEE Data Engineering Bulletin,* Vol.23, N.3, 2000, pp.25-32.

[7] D.Montesi and A.Trombetta. Similarity-based search through fuzzy relational algebra. *Proceedings of 1st Worksop on Similarity Search* (IWOSS'99), Folrence, Italy, 1999.

[8] S.Santini and R.Jain. Similarity Matching, in *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1997.

[9] Special issue on using context in applications, Int. Jour. of Human-Computer Studies,Vol. 48, n.3, 1998.

[10] D.A.White and R.Jainn. Similarity Indexing with SS-trees. *Proceedings 12th IEEE Internat.Conference on Data Engineering*, New Orleans, Louisiana, Feb.1996.

Multimedia Metacomputing[†]

Ulrich Marder Jernej Kovse

University of Kaiserslautern
Dept. of Computer Science
P. O. Box 3049
D-67653 Kaiserslautern
Germany

{marder,kovse}@informatik.uni-kl.de

Abstract

The concept of multimedia metacomputing involves the formation of a large scale loosely coupled multiprocessing environment capable of performing complex transformations on media objects. The transformations are provided in the form of operations integrated in special media processing components. The components are described by signatures that denote the runtime environments required for component deployment, the types of media objects the component operations accept and emit and a formal description of transformations they perform. The multiprocessing environment also connects a set of heterogeneous processing resources in which the components are dynamically deployed in order to carry out the transformations. The existing Internet infrastructure is used to connect storages of media processing components, available processing resources and the system controlling the transformation process. By such an environment, we try to realize the concept of delivering global media data without the need to generate specially adapted materialization of the media data in advance. An open "plugable" environment provides the possibilities for both vendors of media processing components as well as providers of processing resources to exploit the potential of the business model involved in offering and providing multimedia services using the existing Internet infrastructure.

1 Introduction

Over the last couple of years, the Internet has significantly improved in the sense of the variety of different media types it involves. The introduction of complex media types, such as graphics, sound and video clips has made the usage of various Internet services, ranging from electronic mail to the World Wide Web (Web), more appealing. However, the existing Internet infrastructure along with its protocols today still is mainly used to merely support the exchange of media objects. It would be useful if we could find a way to combine this exchange with the possibilities of media processing. This way, the Internet infrastructure would be used to form a large, loosely coupled multiprocessing environment where various kinds of media objects could efficiently be found and processed according to the requirements that may be posed by human as well as certain types of software agents.

[†] This work is supported by the Deutsche Forschungsgemainschaft (DFG) as part of the Sonderforschungsbereich (SFB) 501 "Development of Large Systems with Generic Methods".

During the last decade, metacomputing concepts have been invented to support the dynamic distribution of processing components in high-performance multiprocessing environments. While early approaches were targeted at homogeneous massive parallel systems [9], newer approaches, e. g. [3], often exploit the advantages of distributed component architectures. Our proposal of a multimedia metacomputing environment enhances the component-based approach with dynamic configuration, optimization, and multimedia-specific semantics. In particular, this kind of environment involves the following parts:

- mechanisms supporting storage and retrieval of various types of processing components that enable media objects to be transformed according to specific user requirements,
- processing and communication infrastructure supporting the transfer of media objects between processing resources used to carry out the transformations specified by chosen processing components,
- a special control system supporting scheduling and dynamic migration of processing components between the resources as well as initialization and gathering of the results of the media transformation process taking into consideration the availability and the existing processing load for a certain resource,
- a semantic model supporting the description of multimedia processing tasks independently from concrete processing components, optimization strategies, and materialization of media objects.

In the following, we describe the requirements that each of the parts has to fulfill in order to be able to form the heterogeneous open multimedia metacomputing environment.

2 Component-based Multimedia Metacomputing

2.1 Providing and storing media processing components

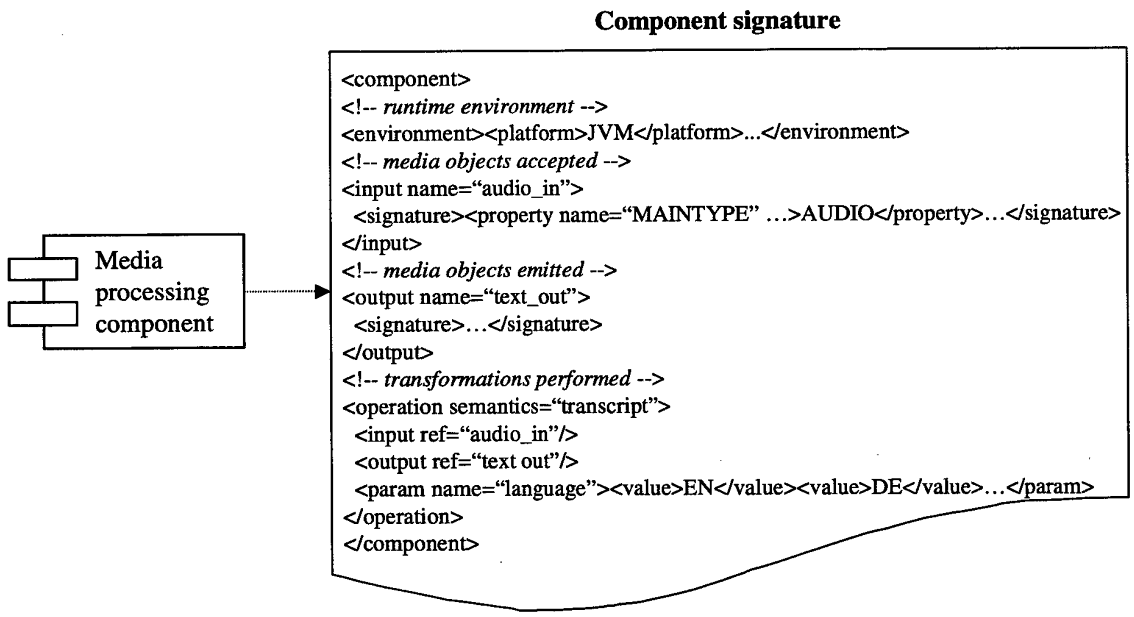
Using the definition provided by the Unified Modeling Language (UML) Specification [7], a component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files. As such, a component may itself conform to and provide the realization of a set of interfaces, which represent services implemented by the elements resident in the component [7]. Over the last couple of years, the so-called Component-based Software Development (CBSD) [1] has become highly popular primarily because of its promise of reducing costs and time needed to produce software products using components as their building blocks. Today, component technologies such as JavaBeans [8] may be used to support the CBSD.

Using the term media processing component, we refer to a software component providing a set of services used to transform the state of a media object. In our case, it is not necessary that such a component conforms to one of the predefined technologies, such as JavaBeans, for example. A component should merely be deployable in the sense of being able to find an appropriate run-time environment tied to a processing resource where the operations provided by a component may actually be applied to a media object. However, an important aspect of using predefined technologies is that the issues of specifying a set of interfaces, component deployment, and component cooperation are already defined, which makes the usage and combination of such components easier.

After a deployed component receives a media object, it applies a sequence of transformations and delivers a transformed object as its output. An audio transcription component, for instance, first performs a speech recognition and then generates a text object containing the transcript. The transformation process is configurable by a set of parameters, which makes it possible for the user agents to influence the process of applying the transformations.

2.1.1 Describing component services

In our scenario, components are stored using a special storage mechanism. In order to be able to successfully locate and retrieve components according to the transformations of the media object that have to be carried out, a component has to provide not only processing functionality, but also a formal description of the transformation process it supports. Also, the types of media objects the transformations may be applied to, have to be precisely specified. This way, the result of the process of applying transformation operations is exactly defined and the control system knows what kind of result a media object transformation delivers. We call such additional information related to transformation functionality a *component signature*. In case a component provides a formal description of its set of interfaces, as it is the case with the majority of common component technologies, the component signature is to be comprehended as an upgrade of such a description that defines not only which operations may be invoked, but also the exact effects of applying the transformation operations to a media object. Component signatures may not be provided directly by media processing components and may therefore be stored separately. Hence, relationships need to be established between storage representations of components and component signatures. Figure 1 illustrates possible usages of provided component signatures.



```
<component>
<!-- runtime environment -->
<environment><platform>JVM</platform>...</environment>
<!-- media objects accepted -->
<input name="audio_in">
  <signature><property name="MAINTYPE" ...>AUDIO</property>...</signature>
</input>
<!-- media objects emitted -->
<output name="text_out">
  <signature>...</signature>
</output>
<!-- transformations performed -->
<operation semantics="transcript">
  <input ref="audio_in"/>
  <output ref="text out"/>
  <param name="language"><value>EN</value><value>DE</value>...</param>
</operation>
</component>
```

Figure 1: Media Processing Component with Sample Component Signature (shortened)

2.1.2 Managing dependencies between component versions

In a lot of cases, not only components, but also relationships declaring dependencies and possibilities of cooperation between them have to be stored and managed. For example, a component declares by its set of interfaces that it is capable of carrying out an operation that performs a media object transformation as required by the user. However, in the course of this transformation, services of another component are required. For this reason, relationships between components have to be established to make the control system aware of this dependency. This makes it possible to successfully deploy both components in appropriate run-time environments, so that the media object transformation can be carried out. For example, the transcription component mentioned earlier could as well be realized as a composition of two other components: speech recognition and text generation. However, the storage mechanism should also be capable of storing and managing different versions of the same media processing components. A new component version may provide improved functionality related to media processing, but may prove to be incompatible with other components the initial component depends on. Therefore, various versions of the same component should be stored and managed by the storage mechanism. Moreover, the relationships between the components that define the dependencies should be refined in such a fashion that it is possible to choose and deploy the appropriate configuration of component versions that fits the desired context of a media object transformation. Hence, following a similar approach as with components, *configuration signatures* are required. Figure 2 illustrates an example of configurations of different component versions. Mahnke et al. [4] describe more general advantages of using customized version control in repositories.

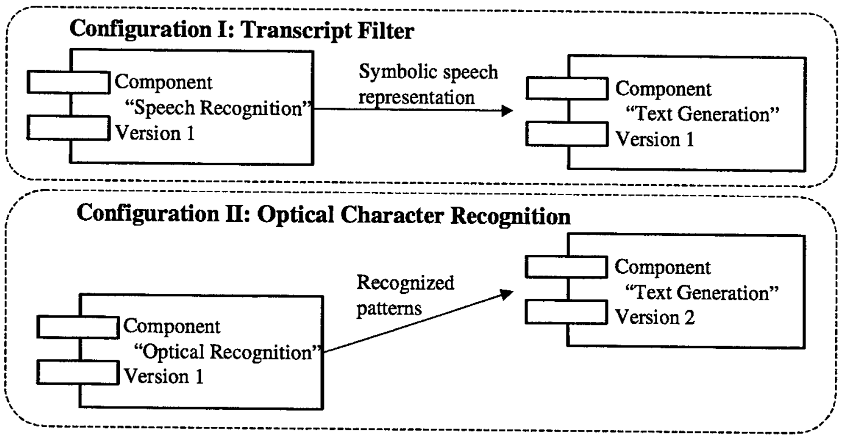

Configuration I: Transcript Filter
Component "Speech Recognition" Version 1
Symbolic speech representation
Component "Text Generation" Version 1
Configuration II: Optical Character Recognition
Component "Optical Recognition" Version 1
Recognized patterns
Component "Text Generation" Version 2

Figure 2: Configuration Example

2.1.3 Storing media processing components

Because of the complex requirements related to storing meta information to support searching of the appropriate components and managing valid configurations of component versions, we think that the services of a file system offer only limited functionality to provide storage facilities for media processing components. Therefore, it is essential to provide storage facilities in a form of special component repositories. Such a repository usually provides standard amenities of a database management system (DBMS), such as data model, query facility, view mechanism, and integrity control. However, in our case the functionality is upgraded using special value-added repository services, such as efficient searching for stored components according to desired functionality as well as version and configuration control.

Note that a repository may be distributed in general such that various media processing components may actually be stored at different locations. Various component vendors may produce components providing media processing functionality, provide component signatures for these components, store them at their own location and register them with the repository. This way, a repository seamlessly integrates various storage locations in a single virtual storage environment, which makes it possible for the control system to find and retrieve the components in a simple fashion (see Figure 3).

2.2 Processing and communication infrastructure

The *processing and communication infrastructure* enables the control system to locate the resources available for media processing, transfer data related to a media object to these

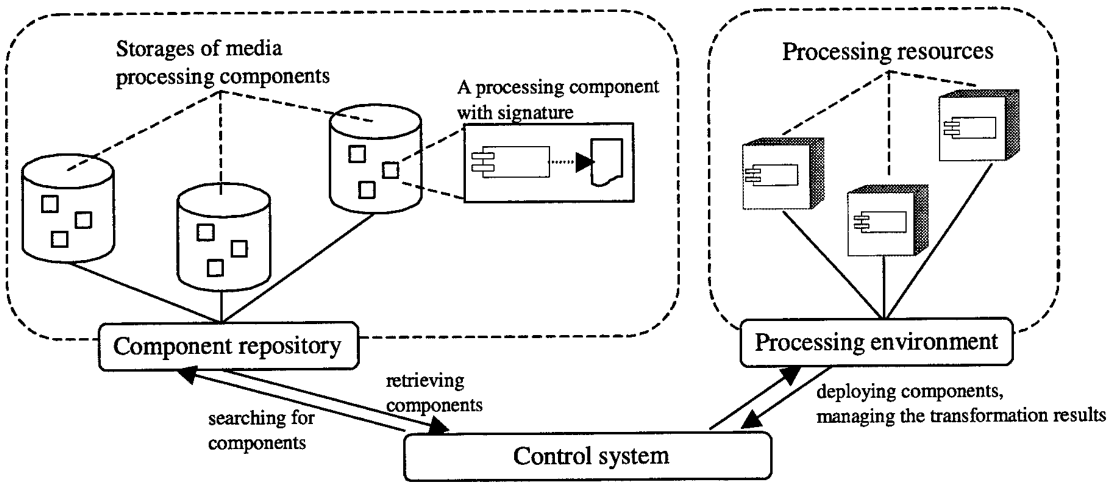

Storages of media processing components
A processing component with signature
Processing resources
Component repository
Processing environment
retrieving components
searching for components
deploying components, managing the transformation results
Control system

Figure 3: General Architecture of the Multimedia Metacomputing Environment

resources and initiate the transformation process that may take place at various resources in a parallel or sequential manner.

Processing resources form special run-time environments in which media processing components are deployed in order to perform transformations on the media objects. Note, since our multiprocessing environment involves multiple types of different processing components, the communication infrastructure links heterogeneous run-time environments. Run-time environments register with the control system, where the exact environment type and the possibilities of component deployment are described.

2.3 Control system

A special *control system* is used to direct the transfer of components to appropriate run-time environments as well as the transfer of media objects between the components on various stages of the performed transformations.

As an input, the control system receives data related to the media object and a formal description of a transformation that has to be performed on the object. The description may be given as a list of commands in a special-purpose high-level language like VMML [6] (cf. Figure 4). The system tries to analyze the description to obtain a list of basic transformation tasks needed. Using special query capabilities of the distributed repository of media processing components, it tries to locate the components capable of performing the requested transformations. In the process of searching for the components, component signatures stored in the repository are used. In case the component is capable of carrying out the transformation, additional information about dependencies of the component to other components may be delivered so that it is possible to choose a valid configuration of cooperating components. Next, run-time environments needed to deploy the components have to be chosen among the environments that have registered with the control system. After component deployment, a media object that needs to be transformed is passed as an argument to the components along with additional arguments used to configure the transformation process performed by the components. The process of deploying appropriate components to run-time environments, configuring them and passing the media object is repeated for each of the basic transformation tasks. As a result of this process, an object transformed according to the user specification is delivered to the user.

Note that due to different types of media processing components, the way of accessing a component by a control system and retrieving the results of the transformation process may vary. In this aspect, the usage of predefined component models proves to be easier, since such models already define the way clients access component services, pass a media object and other configuration parameters, and retrieve the results of the transformation of the media object. However, the usage of other components, such as binary executables deployed in an operation system environment requires additional functionality needed to access the services to enable the communication with the control system. This functionality may be provided by a component vendor as a separate part of the software that is deployed in the same run-time environment and is used as a mediator between the control system and the actual component performing media transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<?doctype vmd system "vmd.dtd"?>
<vmdesc>
  <source>
    <moid alias="bc_video" ext_ref="CNN_db/CNN_Videos/4711"/>
  </source>
  <virtual name="TranscriptedSpeech">
    <signature>
        <property name="Maintype" class="Typespec">Text</property>
        <property name="Subtype" class="Typespec">Plain</property>
        <property name="Encoding" class="Typespec">UTF-8</property>
    </signature>
    <transformation name="transcription">
      <operation semantics="transcript">
        <input alias="i1" ref="bc_video"/>
        <param name="language" value="EN"/>
      </operation>
    </transformation>
  </virtual>
  ...
</vmdesc>
```

Figure 4: Sample Client Request using VirtualMedia Markup Language (VMML)

3 Semantic Model for Multimedia Metacomputing

The component-based metacomputing foundations described in the previous section form a necessary prerequisite for multimedia metacomputing. We need, however, also a semantic model clearly specifying

- an abstract collaboration model,
- the external representation of media objects, operations, and client requests, and
- how client requests are pre-processed (transformed) to become executable by the metacomputing environment.

This model supplies the user or application programmer with everything needed to create both ad-hoc requests and metaprograms (e. g., request templates). The model also precisely describes how these requests get transformed into plans executable within a given metacomputing environment. The advantages of letting users make requests instead of directly creating plans are:

Ease of use: Requests are much simpler to create, because one does not have to deal with finding components implementing a certain operation, manipulating the media objects in order to fit them to the selected operation's signature, and so on.

Stableness: Requests are stable while plans are not. The reason is the inherent unstableness of a Web-based metacomputing environment, in which resources may become unavailable, replaced, or updated from time to time. A plan fails, if one of the required resources is not available or compatible anymore, whereas a request would result in an alternative plan (if one exists).

Optimization: A request may be transformed into different plans depending on volatile conditions. We may, for example, consider time constraints, cost limits, utilization of resources, and exploitation of redundancy.

Due to space limitations, we only sketch the major aspects of the model in the following sections.

3.1 Abstract Collaboration Model

Our collaboration model is based on filter graphs (cf. Figure 5) similar to those introduced in [2]. The start nodes of the graph are media producers p_i (media objects managed by some server, maybe even live media sources) and the end nodes are media consumers c_i (e. g., client applications). The intermediate nodes are media filters f_i, the basic operations of a media computation, while the edges of the graph represent media streams flowing from one filter (or media producer) to another filter (or media consumer).

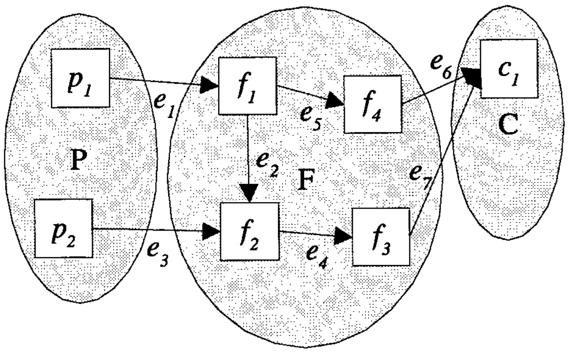


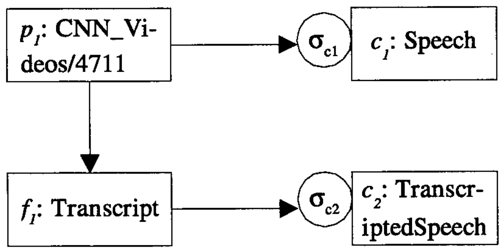
Figure 5: Illustration of the Collaboration Model

The graph in Figure 5 can be interpreted as follows. There is a producer p_1 creating a media object sent to filter f_1. Filter f_1 generates from its input two media objects which are sent to filters f_2 and f_4, respectively, and so on. Thus, the graph nodes are not bound to any real instances of media servers or filters, therefore we call this an abstract collaboration model.

3.2 Abstract Media Semantics

Turning our collaboration model into a multimedia metacomputing model requires the addition of some more specific media semantics to the graphs. This is done by means of signatures. A signature is a set of properties belonging to any of the following property categories: type, quality, content, functional, and non-functional specifications.

Figure 6 shows an example of a filter graph with signatures, in which the path $p_1 \rightarrow f_1 \rightarrow c_2$ corresponds to the sample request presented previously in Figure 4. The rectangular nodes contain node signatures. An edge has two signatures, one for each end. Edge signatures may be empty. In Figure 6, non-empty edge signatures are drawn as a circle between edge and node.



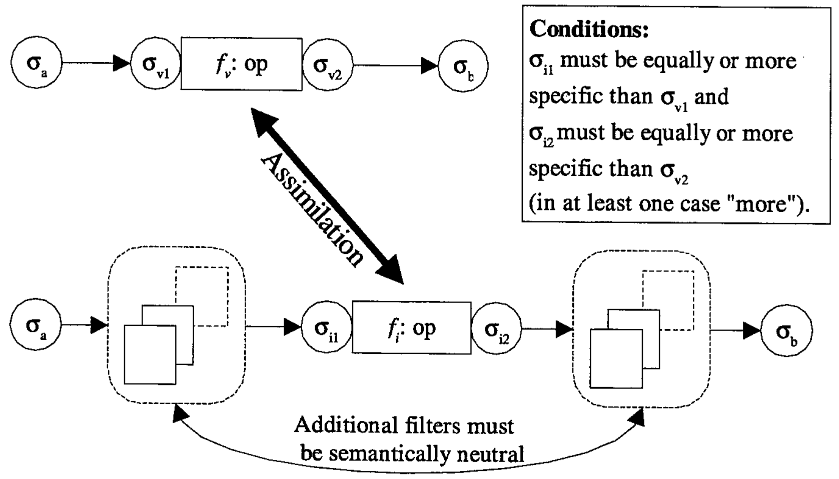
σ_{c1}:	σ_{c2}:
[Typespec]	[Typespec]
Maintype=Audio	Maintype=Text
Subtype=Waveform	Subtype=Plain
Encoding=WAV	Encoding=UTF-8
[Quality]	
Sampling_Frequency=44100	
Sample_Depth=16	

Figure 6: Sample Request Graph with Signatures

A graph like the one in Figure 6 describes the computation of some media objects on a very abstract level, yet reflecting the full semantics from the client perspective.

3.3 Request Processing

The first thing to note is that the addition of edge signatures may turn the edges into a sort of "magic channels". Consider, for example, the edge between p_1 and c_1 in Figure 6. The signature σ_{c1} contains some type properties of the media object emitted by this channel, but what goes into this channel fully depends on the internal representation of the media object referenced by p_1. Thus, we can get edges with incompatible signatures at both ends. Apparently, there should be some kind of hidden computation within the channel, hence, the name "magic channel".



σ_a
σ_{v1}
f_v: op
σ_{v2}
σ_b
Conditions:
σ_{i1} must be equally or more specific than σ_{v1} and
σ_{i2} must be equally or more specific than σ_{v2}
(in at least one case "more").
Assimilation
σ_a
σ_{i1}
f_i: op
σ_{i2}
σ_b
Additional filters must be semantically neutral

Figure 7: Equivalence of Abstract Operations (Filters) and (possible) Implementations defined as Semantic Assimilation

Consequently, one of the major purposes of request processing is revealing the semantics of "magic channels" in such a way that the request becomes executable by configuring and integrating appropriate components. In other words, the request graph has to be transformed into a semantically equivalent graph that contains only deployable components as nodes and edges with compatible media signatures at both ends. This automatic process has to be supported by formally specified semantic equivalence relations.

The three basic equivalence relations are *neutrality*, *reversibility*, and *permutability*, thus defining which media operations are considered semantically neutral, which operation is the inverse of another, and which groups of operations are semantically independent from each other, respectively. The semantics of media *composition* and *decomposition* is defined by generalized forms of these relations. The most important equivalence relation is called *semantic assimilation* (see Figure 7), which defines the relation between an abstract media operation (usually specified in a user request) and a component or configuration of components implementing that operation. More detailed explanations and a discussion of other important issues like, for instance, materialization management can be found in [5,6].

4 Conclusions

In this paper, we described the so-called multimedia metacomputing approach that aims at the formation of a large scale loosely coupled multiprocessing environment providing a distributed architecture to perform transformations on media objects. Basically, the following conclusions emerge from our previous discussion:

- operations that perform the transformations on media objects can be provided in the form of special media processing components,

- each media processing component should provide a signature to formally describe the run-time environment it requires during its deployment, types of media objects it accepts and the transformations it performs,
- it proves to be essential to exploit services of a repository as a distributed storage mechanism for processing components; in comparison to other solutions, a repository may provide additional functionality related to component versioning as well as combining component versions into valid configurations capable of cooperation in the process of media object transformation,
- an abstract semantic model has to be provided to ensure (semantically) correct request processing and robustness of client programs against changes of the metacomputing environment like, for example, exchange of components, processors, or media object materialization.

A global multimedia metacomputing environment would, in principle, allow to deliver global media data to any client and any kind of multimedia device without need to generate especially adapted materialization of the media data in advance. Moreover, computationally complex transformations and manipulations of the data are dynamically delegated to the most appropriate processing resources at run-time, thus optimizing response time and utilization of expensive special-purpose hardware. Ultimately, a "plugable" model for vendors of components providing media transformations and processing resource providers could form the (technical) foundation of a flexible business model for offering and vending multimedia services over the Internet. Albeit, our future work will be related to:

- further exploring the possibilities of using existing component technologies, such as JavaBeans in our approach,
- developing a mechanism used to dynamically evaluate the performance of processing components deployed in run-time environments while carrying out transformations on the media objects; using this mechanism, the results obtained are stored in a special history database and later used by the control system in order to achieve improved response times in the instances of subsequent media transformations.

References

1. Brown, A. W.: Large Scale Component Based Development, Prentice Hall, 2000.
2. Candan, K. S., Subrahmanian, V. S., Venkat Rangan, P.: Towards a Theory of Collaborative Multimedia. In: Proc. IEEE International Conference on Multimedia Computing and Systems (Hiroshima, Japan, June 96), 1996, pp. 279–282.
3. Hawick, K. A., James, H. A., Silis, A. J., et al.: DISCWorld: An Environment for Service-Based Metacomputing. In: Future Generation Computer Systems, 15 (5–6), 1999, pp. 623–635.
4. Mahnke, W., Ritter, N., Steiert, H.-P.: Towards Generating Object-Relational Software Engineering Repositories. In: Proc. 8th GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", BTW '99 (Freiburg, Germany, March 1–3), Buchmann, A. (ed.), Informatik aktuell, Springer-Verlag, March 1999, pp. 251-270.
5. Marder, U.: On Realizing Transformation Independence in Open, Distributed Multimedia Information Systems. In: Proc. 9th GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", BTW '2001 (Oldenburg, Germany, March 7–9), Heuer, A., Leymann, F., Priebe, D. (eds.), Springer-Verlag, Heidelberg, Berlin, March 2001, pp. 424–433.
6. Marder, U.: Transformation Independence in Multimedia Database Systems. SFB-Report 11/2000, SFB 501, University of Kaiserslautern, Nov. 2000, 24 pages.
7. OMG, Unified Modeling Language Specification, version 1.3, OMG Document ad/00-03-01, March 2000.
8. Roman, E.: Mastering Enterprise JavaBeans, John Wiley and Sons, 1999.
9. Smarr, L., Catlett, C. E.: Metacomputing. In: Comm. ACM, Vol. 35 No. 6, June 1992, pp. 44–52.

A system for Query By Example in Image Data Base

Angelo Chianese Antonio Picariello Lucio Sansone

Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II
via Claudio, 21. 80100 Napoli ITALY
{angelo.chianese, picus, sansone}@unina.it

ABSTRACT

Processing queries for databases containing multimedia information is a hard question.

In this paper we describe a knowledge base approach to effectively resolve query by examples in image database. We present a prototype system that is at the moment under developing in our Labs. We consider wavelet descriptors for modeling color, shape and texture features, and we propose a knowledge base for assisting the user in the retrieval process.

Results on a collection of about 1000 images are reported to provide a validation of the proposed strategy.

Keywords
Image Database, Content Based Image Retrieval, Data Mining, Multimedia Query Processing.

1. INTRODUCTION

The development of the world wide web and fast computer technology are at the origin of the large amount of visual information retrieval systems: the growing needs of how to assist a user in accessing and retrieving images has become an important problem [1]. Traditional multimedia DBMS use textual keywords as an index to quickly access to multimedia data. However, this kind of representation requires a burdening manual processing.

Query By Example (QBE) is considered a promising approach because an user handles an intuitive query representation, that is an image itself. Clearly, the greatest difficult is to find *feature vectors* that effectively represent an image[2].

In a QBE system the basic question is "when two images may be considered similar". In the literature, similarity of images has been expressed using four features: colors, texture, shape and spatial position [3].

Several prototypes [4], [5], [6], [7] and commercial systems [8], [9] have been proposed. The techniques, however, can be more or less summarized as follows:

1) a feature extraction module – based on Computer Vision techniques – is used to detect prominent features from image – or more generally multimedia – objects;
2) a similarity concept – based on a distance metric – is determined;
3) the retrieval process is performed by means of a similarity distance in the feature space.

However, it is evident that the effective use of such techniques in a real Data Base, requires the development of strategies for mapping high-level features to low level visual descriptors. According to [11] – and it is also the author's opinion –Knowledge Discovery (KD) techniques are suitable to automatically reach this goal.

In the literature, KD refers to non trivial extraction of implicit, previously unknown and potentially useful information from the data stored in a certain database [12]. The use of KD techniques for large data base has been used extensively in the last decade [13], [14], [15], [16], [17], expecially exploiting the concept related to learning from examples techniques [18].Several system prototypes have been released, as INLEN [17], KDW+ [19], Quest [13], IMACS [16], Datalogic/R [20], 49er [21]. Few systems have been specialized for multimedia related problems.

Paper originality consists of proposing a new approach to image query processing that exploits the rules, extracted from visual low level descriptors, by means of a KD module, and produces a Knowledge Base useful for query rewriting and pre-processing. Innovative visual descriptors based on Wavelet and a novel KD module have been used and integrated in the developed system.

The paper is organized as follows: first we provide a description of the system architecture. Section 3 is dedicated to a detailed description of the proposed strategy, describing the features, the KD module and the problems related to a Fuzzy Knowledge Base (FKB) in order to produce a semantic catalogues for fast image retrieval. A discussion of experiments and results is also provided.

2. SYSTEM ARCHITECTURE

It is well known that traditional Data Base Management System (DBMS) are not suitable for managing multimedia data.

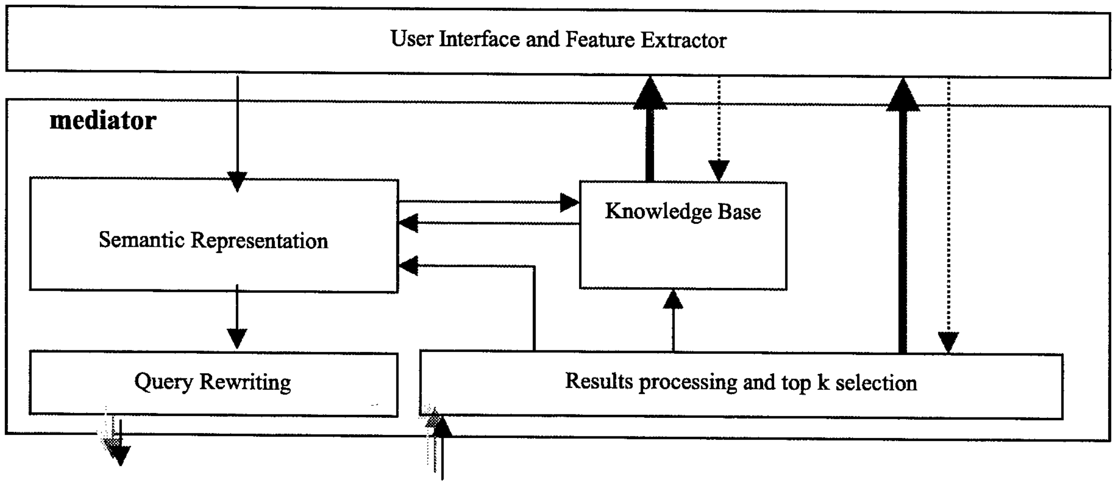

User Interface and Feature Extractor
mediator
Semantic Representation
Knowledge Base
Query Rewriting
Results processing and top k selection

Figure 1: System Architecture

In our work, we used a *mediator based architecture* [22] to manage the complexity related to multimedia data.

Our mediator serves as an interface between the user and the information sources. Its main task is to decompose a complex query into a sequence of more simple and efficient sub-queries.

In our work, the mediator uses a set of abstract properties, that have been extracted in an automatic or semi automatic way.

Figure 1 outlines a synthetic description of the system, that is actually being developed in the Multimedia Database Lab of our Department.

An image is decomposed into a set of features (*physical descriptors*) that are analyzed by a Fuzzy Knowledge Base (KB). The FKB is used to transform the queries based on visual descriptors into a set of *symbolic* queries and the results are finally analyzed by a *"top-k selection"* stage.

In the following we describe the Data Model and the Fuzzy Knowledge Base.

3. DATA MODEL

Image Data Bases (IDB) are typically examples of available information in a descriptive and imprecise form that requires a fuzzy representation.

A fuzzy set [23] consists of data and their corresponding grades of memberships in the set.

Let a fuzzy relational schema be a collection of rules in the form R: $\{A_1, A_2, , A_n\}$ where $A_i \subseteq D_i$, D_i being the fuzzy domain from which A_i are selected.

An instance of relation schema R denoted by r consists of values of attributes A_i.

A tuple of R is formed by the elements $v_1, v_2, \dots, v_n$, elements, $v_i \in A_i$. Each component v_i is atomic or a set value.

For the aims of image databases, we model the values $v_i \in A_i$ as

$$v_i = \{(a_{i,1}, p_{i,1}), (a_{i,2}, p_{i,2}), \dots, (a_{i,K}, p_{i,K})\} = t[A_i]$$

where a_i is an atomic value and $p_{i,j} = \mu_A (a_{i,j})$ denote the membership function.

The uncertainty of any value is defined as fuzzy predicate, and expressed by $p_{i,j} \in [0,1]$.

The value $p_{i,j}$ has the following meaning

1) $p_{i,i} = 0$ means that the value is completely false;
2) $p_{i,j} = 1$ means that the value is completely true;
3) $0 < p_{i,j} < 1$ means that the value is true to the degree expressed by the real number $p_{i,j}$..

Example:

A simple fuzzy relation r according to our model is showed as follows.

File	Color	Shape	Content
Im1.gif	{(red, 0.78), (grey, 0.80)}	{(circle, 0.79), (ellipsis, 0.89)}	{(Sun in the dark, 0.7)}
Im2.gif	{(blue, 0.89), (black, 0.90)}	{(rectangle, 0.94), (square, 0.90)}	{(Sea, 0.8)}

Suppose that t_1 represents the first tuple listed above. This tuple may be interpreted as saying that:

t_1.File = Im1.gif with certainty equal to 1;

t_1.Color = red with certainty 0.78 and t_1.Color = grey with certainty 0.80;

t_1.Shape = circle with certainty 0.79 and t_1.Shape = ellipse with certainty 0.89;

t_1.Content = Sun in the dark with certainty 0.7;

A Fuzzy Knowledge Base (FKB) [24] is made up of a set of rules a set or rules $Rx = \{R^1, R^2, \ldots R^S\}$, where each R^i is in the form:

$$R^i = (A^1, \mu_A{}^1) \ \Phi \ (A^2, \mu_A{}^2) \ldots. \ \Phi \ (An, \mu_A{}^n) \rightarrow (C, \mu_C)$$

A^i being a fuzzy subset having a grade $\mu_A{}^i$, and C being a fuzzy subset having a grade μ_C and Φ being a fuzzy operator *and, or* and *not.*

Our proposed FKB is formed by a set of rules described as follows.

Let an object be a region of an image selected by a user by means of the visual interface module. Over the object domain, we define two classes of rules:

Simple rules. Simple rules describe the visual descriptors of an object, i.e. shape (circle, rectangle, lines, and so on) texture, colors (red, yellow, and so on), position (North, West, South, East, NorthEast and son on). Let f_i be defined in an interval set $[f_{ia}, f_{ib}]$. We define *simple rule* a rule in the form:

$$f_1 \in [f_{1a}, f_{1b}] \ \Phi .. \ \Phi f_n \in [f_{na}, f_{nb}] \rightarrow basicFeature = \text{'bF1'}, p_{bf1}$$

where basicFeature is one of the attribute {shape, colors, texture, position} bF1 is a value in the attribute Domain, p_{bf1} is the grade of confidence of the rule. Note that the previous rule is a fuzzy rule having all the antecedents grade equal to 1.

Complex rules. Complex rules describe the concepts relating to a certain image I. In our approach the symbolic visual descriptors are combined to produce a rule which describes the image content at a higher level. A complex rule is as defined as a rule in the form:

$$(bF1, p_{bf1}) \ \Phi \ldots \ \Phi \ (bFN, p_{bfN}) \rightarrow highLevelFeature = \text{'hlf'}, p_{hlf}$$

where basicFeature, bF_i and p_{bf1} have the same role played in the simple rules, and highLevelFeature, 'hlf', p_{hlf} refer to concepts.

4. IMPLEMENTATION

A central role of our methodology is played by the FKB. It is the authors' opinion that the ability to perform advanced "query by examples" based on similarity criteria can be enhanced by means of the FKB itself

The basic question is so how to build a FKB for a given IDB. This task is performed by a Knowledge Discovery module.

Figure 2 shows the steps we have used for obtaining the FKB rules .

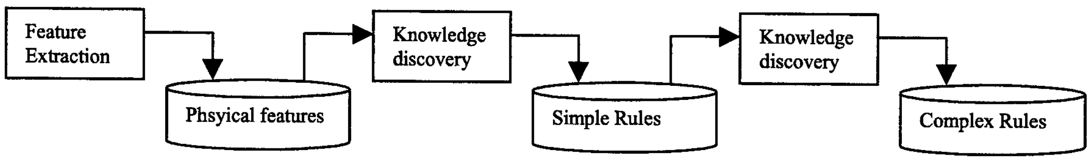


Figure 2: FKB Building

The KD module analyzes the structural features of an image and produces a first set of rules. The structural – or "physical" – features describe colors, texture and shape and they are extracted by the *feature extraction* layer.

We have chosen a certain number of samples from the given IDB in order to build a *training set* of features organized into a "feature vector". We have associated a *symbolic* description – determined by a human – to each feature vector (*supervised learning*). The feature vectors and each associated symbolic description is analyzed by the KD module in order to extract a set of *simple* rules. Simple rules are organized and associated to a high level description provided by humans again. In this way, the KD modules produces a set of *complex* rules.

In other words, the KD module performs a classification task. KD has been implemented in one of the author's previous work [25] and is based on genetic algorithms.

At the end of the classification process, the FKB contains two kind of classes:

a) the ones related to the considered visual descriptor: classes of textures, colors, shapes and spatial position;

b) the others related to the content of images or regions of an image (objects into image) contained into the given database: sea, landscapes, city skylines, animals and so on.

The KD module associates a reference class to each discovered rule and a *membership grade.* We assumed that the nature of the extracted rules is *fuzzy*, in order to manage the uncertainty related to the similarity concepts between two images: however, we also note that several model dealing with uncertainty are proposed by using the fuzzy set theory [26].

In the following, we show the descriptors we have used in our work.

4.1 PHYSICAL FEATURES

It is well known in the literature that the similarity of two images may be expressed using colors, texture, shape and spatial descriptors. The features used in this paper are an extension of those proposed in [27]: at the moment, only *color*, *texture* and *shape* features have been considered

Color is described using *color histograms* calculated on the sub-sampled, low-pass, image in the HSV color space [27].

Texture features are provided by the wavelet covariance signature [28]. In particular, we define the wavelet covariance as:

$$C_{ni}^{X_j,X_i} = \int I_{D_{ni}}^{X_j}(\vec{b}) I_{D_{ni}}^{X_i}(\vec{b}) d\vec{b}$$

I_D^X being the detail subband of the X component in a given Color Space of an Image I, $\vec{b}$ ranging over the domain of the subbands.

Let us call the set

$$\{C_{ni}^{X_j,X_i}\}_{n=0,\ldots,d-1;i=1,2,3}^{j,k=1,2,3;j\leq k}$$

the wavelet covariance signature.

The shape features are calculated using a shape-from-texture algorithm [29]. The algorithm uses Gabor wavelets for the extraction of local spectral moments, considering as features the second order moments of the local surface spectra:

$$a_s(x_s,y_s) = \iint u_s^{2} W_a I(x_s,y_s;x_s,y_s) du_s dv_s$$

$$b_s(x_s,y_s) = \iint 2u_s v_s W_a I(x_s,y_s;x_s,y_s) du_s dv_s$$

$$c_s(x_s,y_s) = \iint v_s^{2} W_a I(x_s,y_s;x_s,y_s) du_s dv_s$$

The above descriptors are organized into a feature vector $\underline{\mathbf{F}}_V$ formed by 78 fields, containing
Reduced HSV histogram description *(field0-field48)*
Texture Code *(field49-field75)*
Shape Description *(field76-field78)*

4.2 LOW LEVEL DESCRIPTORS

$\underline{\mathbf{F}}_V$ is associated to a symbolic description provided by humans.

This stage is hardly dependant on the given image database. In our experiments, we have used the following descriptors.

Feature	**Symbolic Description**
Color	black, white, green, red, yellow, brown, orange, blue, blue light, purple, violet, gray, pink, red dark, beige
Texture	thin_texture, thick_texture, mixted_texture, plated_texture, rectangular_thin_texture, rectangular_thick_tecture, cresped_texture
Shape:	triangle, rectangle, circle, undefined shape

Examples of *simple rules* that are extracted by KD is given in the following table

```
[field27]>2 AND [field27]<67 AND [field28]<99→bleu, 0.653
[field15]<90 AND [field25]>8 AND [field25]<99→bleu, 0.827
[field4]<22 AND [field11]<99→green, 0.96154
[field34]<95 →green, 0.816
field12]>1 AND [field12]<58→white, 0,918
[field11]>1 AND [field11]<15 →white, 0,939
```

The numeric value which follows the symbolic description represents a fuzzy membership grade.

4.3 HIGH LEVEL DESCRIPTORS

The set of simple rules is analyzed by KD module once again, and a High Level Descriptor is associated. to each rules

In the following we show the used descriptors.

Feature	**High Level Symbolic Descriptors**
HighLevel	Landscape, snowy landscape, mountain landscape, coast landscape, waterfall landscape, sunset landscape.

The following example shows some high level rules from our FKB:

```
[white]>0,918 AND [gray]>0,561 AND [mixt_texture]>0,674 → SnowMountainPicture,  0.918
[purple]>0,856 AND [bordeaux]>0,765 AND [yellow]>0,449 AND [thin_texture]<0,704→SunSetPicture, 0.839
[bleu]>0.827 AND [beige]>0.765 AND [green]>0.449 AND [thin_texture]<0.704 →SeaCoastPicture, 0.939
```

For the sake of clarity, we show the results of an analysis performed over a given image.

Let us consider the following sample image.

Our FKB produces the following rules:

```
[bleu]>0.827 AND [beige]>0.765 AND [green]>0.449 AND [thin_texture]<0.704 →SeaCoastPicture, 0.939
[field27]>2 AND [field27]<67 AND [field28]<99→bleu, 0.653
[field15]<90 AND [field25]>8 AND [field25]<99→bleu, 0.827
[field4]<22 AND [field11]<99→green, 96.154
[field34]<95 →green, 0.816
[field75]>-17105,29054 AND [field75]<34) → thin_texture, 0.633
[field55]>92 AND [field55]<1814756,920325 → thin_texture, 0.704
```

Once we have built the FKB, the overall IDB is indexed by the FKB itself and the extracted knowledge is stored into the following image catalogue.

ImageCatalogue {***Image*** (ImageID, ImageName), ***Object*** (ImageID, ObID), ***ObjDescription*** (ObID, FeatureID, Confidence), ***Content*** (FeatureID, FeatureValue), ***featDescription*** (FeatureID, FeatureName) }.

The table ***Image*** contains information about the name of the image; the table ***Object*** describes the selected objects for each image; the table ***ObjDescription*** describes the features used for analyzing the object and the probability of positive classification (confidence); the table ***Content*** contains the name of the recognized features, as classified by the use of the KB and of the KD module; the table ***featureDescription*** contains the set of the all considered and analyzed features.

4.4 PROCESSING STAGE

The mediator finds – for each query – a set of equivalent subqueries in order to process QBE queries: an image is processed by the visual interface and the extracted features are analyzed in terms of the FKB. In this way, the set of features derives a fuzzy query Q that is used for query processing. Eventually, the user is provided with a list of several results that satisfy in a certain grade and order the expressed similarity query

For the aims of our work, a query Q is decomposed into a set of subqueries, say Q= {q1, q2, ..., qn}.

We assumes that:

(i) each subquery is a query expressed in terms of symbolic values and confidence (grade of membership).

(ii) the result of each query is a graded (fuzzy) set:

$$q_i \rightarrow FR_i = \{(r_{i1}, a_{i1}), (r_{i2}, a_{i2}), ..., (r_{ik}, a_{ik})\}$$

(iii) the computation of q_i may be (or may not be in the case of independence) conditioned by FR_{i-1}.

From a general standpoint, two generic query approaches to query processing are either based on entirely sequential or entirely parallel processing plans. Sequential processing requires a subquery qi to be submitted to a local system if and only if the results of subqueries q1, q2, ..., qi-1 are already available. This method is efficient when the partial results r1, r2, ..., ri-1 can be used to reduce the processing time of the remaining subqueries. Parallel processing plan requires simultaneous submission of all subqueries to the local queries. It is beneficial when the computational complexity of all subqueries is more or less the same and the majority of subqueries cannot be simplified by the results of the other subqueries. In complex similarity queries, we proceed starting from the idea that some of the subqueries are submitted and usefully processed sequentially, while the other ones may be processed in parallel.

To best understand the task performed at this stage, let us consider the following example.

Example: Let us consider that the query rewriting module receives in input the tuple (f_{color} = [a,b], $f_{texture}$ = [v,z]). Let us suppose that the KB module could satisfy a certain number of rules, and determine that the image is similar to those images representing the 'sea' with a confidence value equals to 0.8.

So the query may be rewritten in the form:

select ImageName
from Image **natural join** Object **natural join** objDescription **natural join** Content **natural join** featDescription
where featDescription.name = 'content' and featureValue = 'sea' and objDescription.confidence > 0.8

The nature of the select varies according to the kind of subqueries.

We have considered the following classes of subqueries.

a) query containing low level descriptors
b) query containing high level descriptors
c) a combination of a) and b).

The queries containing high level concepts have been considered independent.

The queries containing visual descriptors may be dependent, and the dependencies may be resolved at processing time.

For our aims, a query is viewed as a query tree whose leaves correspond to a single symbolic query. The nodes correspond to fuzzy operator. Each leaf node in the query tree corresponds to a selection operation through symbolic features on the described catalogue.

The adopted techniques are based on the different kinds of queries as described by a) b) and c), and are inspired to [30].

The global query Q is partitioned into two subset of queries, Q_i and Q_d, where Q_i indicates the queries described in a) and Q_d those of topics b) and c):

$Q = \{Q_i, Q_d\}$

Each node of Q_i and Q_d is analyzed according to the algorithms showed in the following code, through the *independentQueries()* and the *dependantQueries()* procedures appropriately on each node.

Figure 3 shows a description of the implemented procedures.

```
Procedure independentQueries();
Begin
        For each $q_i$ find all the images which are
        similar to $q_i$ and store the result into
        result tables.
        For each couple of result tables, apply
        the fuzzy operators and, or, not,
        according to the original query.
        Show the top k in the final result table.
End;

Procedure DependantQueries();
Begin
        While   there exist queries to process
        Do begin
                $table_1$:= result($q_i$)
                $table_2$ := result ($q_{i+1}$)
                For each element of $table_1$
                        Evaluate the grade of
                        similarity of the given
                        elements with all the
                        elements in $table_2$, using
                        min, max or not (fuzzy).
                End;
                Show the top k in the final result table
End;

Function result(q: query ): table;
Begin
        This function performs the query q with
        confidence p, calling the SQL statement:

        select  ImageName, confidence
        from    Image natural join Object
                natural join objDescription
                natural join Content natural
                join featDescription
        where   featDescription.name =
                "symbolic feature"
                objDescription.confidence > p
end;
```

Figure 3: The independentQueries() and dependantQueries() procedures

5. RESULTS AND DISCUSSION

Preliminary experiments have been conducted over a variety of data sets to measure the performance of the methods and the query processing algorithms developed.

Note that the evaluation of visual perception of similar colored images is a complex process. In the literature no method for measuring effectiveness and efficiency of a query has been found. However, it is largely agreed that the efficiency of an approach may be evaluated as respect to *human* evaluation. Our results have been reported in term of True Positive TP (the percentage of right similar images) and False Positive FP (the number of false detected similar images). The plot of TP vs FP is also called ROC. We have plotted ROC curve as a function of the threshold used in the KD module for deciding the region of confidence of a certain rule.

The experiments have been done on about 1000 images picked out the Internet and several commercial archives. They represent landscape, common objects and pictures. The images have been grouped and stored into a commercial object relational DBMS. About two hundreds images have been used for training building the FKB and a semantic catalogue has been built.

For experiments, we have used Burt and Adelson wavelet [31] and four decomposition levels, 16x8x8 colors and the space color HSV.

The following figure 3 represents the results of the experiments over our set of images.

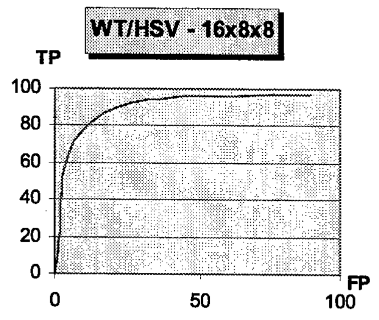


Figure 3: ROC curve

6. CONCLUSIONS AND FURTHER WORKS

To address the needs of application based on Query By Content in image database, we are developing a system based on the integration of computer vision and knowledge discovery techniques.

Our results, reported in terms of ROC curves, show that this method is a promising approach.

However, we note that there are a number of issues that need to be addressed:

- the notion of query dependency has to be more deeply exploited. We have proposed of decomposing a query into either dependent or independent subqueries: we haven't yet discussed how to aggregate the results of these subqueries, which can affect the outcome of the execution (i.e. the top k selection module);
- a study of the computational cost of the proposed strategy;
- at the moment, we do not employ any automatic segmentation techniques. This approach is not scalable to support large image databases: for this reason we are extending the work with an automatic segmentation module based on active vision techniques.

- eventually, we will provide the work with a comparison against other techniques.

7. REFERENCES

[1] A. Yoshitaka, T. Ichikawa, A survey on content based retrieval for Multimedia Databases, IEEE Trans. On Knowledge and Data Engineering, vol. 11, n. 1, 1999, pp.81-93.

[2] E. Albuz, E. Kocalar, A. Khohkar, Vector-Wavelet based scalable indexing and retrieval system for large color image archives, Proc. Of IEEE ICASSP 99, pp. 3021-3024.

[3] T. Caelli, D. Reye, On the classification of Image Regions by colour, texture and shape, Pattern Recognition, vol. 26, n. 4, 1993, pp. 461-470.

[4] J. R. Smith, S. F. Chang, Tools and Techniques for Color Image Retrieval, Proc. IS&T, Storage and Retrieval for Image and Video Databases, vol.2, 670, SPIE 1994

[5] A. Pentland, R.W. Picard, S. Sclaroff, Photobook: tools for Content-Based Manipulation of Image Databases", vol.2, pp. 34-47, SPIE, Bellingham, WA, 1994.

[6] B. S. Manjunath, W.Y. Ma, Texture Feature for Browsing and Retrieval of Image Data, Tech, Report TR-95-06, 1995

[7] S. Mehrotra et al., Towards extending information retrieval techniques for multimedia retrieval, proc. Third Int. Workshop on Multimedia Information Systems, Como, 1997.

[8] M. Flickner et al., Query By Image and Video Content: the QBIC system, IEEE Computer, vol. 28, n. 9, 1995, pp. 23-32.

[9] J. R. Bach et al., The Virage Image Search Engine: an open framework for image management, Proc. SPIE Storage and Retrieval for still

[10] M. J. Swain, D. H. Ballard, Color Indexing, Int. Journal of Computer Vision, vol. 7, n. 1, 1991, pp. 11-32.

[11] C. Djeraba, When image indexing meets knowledge discovery, Proc. of ACM MDM/KDD 2000, Simoff and Zaiane (ed.), pp.73-82

[12] J. Han, Y. Huang, N. Cercone, Y Fu, Intelligent Query Answering by Knowledge Discovery Techniques, IEEE Transactions on Knowledge and Data Engineering, vol. 8, 3, June 1996, pp.373-390.

[13] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large Database, Proc. 1993 ACM-SIGMOD Int. Conference on Managment of Data, pp. 207-216, Washington DC, 1993.

[14] W. J. Frawley, G. Piatetsky-Shapiro, C. J. Matheus, Knowledge discovery in Database: an overview, pp. 1-27, AAAI/MIT Press 1991.

[15] J. Han, Y. Cai, N. Cercone, Data Driver Discovery of quantitative rules in relational databases, IEEE Transaction on Knowledge and Data Engineering, vol. 5, pp. 29-40, 1993.

[16] R. J. Brachman, Viewing data from Knowledge representation lens, Proc. Int. Conference Building and Sharing of Very Large Scale Knowledge Bases, pp- 117-120, 1993.

[17] R. S: Michalsky et al.,, Mining for knowledge in database: the INLEN Architecture, initial implementation and first results, Journal of Information Systems, vol. 1, pp. 85-114, 1992.

[18] J. Han, Y. Fu, Exploration of the Power of attribute-oriented induction in data mining, Advances in Knowledge Discovery and Data Mining, pp. 399-421, AAII/MIT Press, 1996.

[19] G. Platetesky-Shapiro, C. J. Matheus, Knowledge Discovery Workbench for Exploring Business Databases, International Journal Intelligent Systems, vol. 7, pp. 675-786, 1992.

[20] W. Ziarko, R. Golan, D. Edwards, An application of Datalogic/R Knowledge Discovery Tool to identify Strong Predictive Rules in stock market data, Pro. AAAI-93 Workshop on Knowledge Discovery in Databases, pp. 93-101, Washington DC, 1993

[21] J. Zykow, J. Baker, Interactive Mining of regularities in Databases, Knowledge Discovery in Databases, AAAI/MIT Press, 1991

[22] A. Brink, S. Marcus and V.S. Subrahmanian. Heterogeneous Multimedia Reasoning. IEEE Computer, 28, 9, pps 33--39, Sep. 1995.

[23] L. Zadeh, Quantitative Fuzzy Semantics, Information Sciences 3, pp. 159-176, 1971

[24] D. Dubois, H. Prade, L. Ughetto, Checking the coherence and redundancy of Fuzzy Knowledge Base, IEEE Trans. On Fuzzy Systems, vol. 5, 3,pp. 398-420, 1997.

[25] I. Finizio, A system for Knowledge Discovery in Data Base, (In italian), Laurea Degree Dissertation, L. Sansone (advisor), june 2000, University of Naples "Federico II".

[26] R.Fagin, Combining fuzzy information from multiple systems, J. Computer and Systems Sciences, 68, pp. 83-99, 1999.

[27] A. Chianese, G. Boccignone and A. Picariello, Wavelet transform and image retrieval, preliminary experiments, Int. Conference on Computer Vision, Pattern Recognition and Image Processing, Atlantic City 2000

[28] G. Van de Wouwer et al., Wavelet correlation signatures for color texture characterization, Pattern Recognition, vol. 32, 1999, pp. 443-451.

[29] B. J. Super, A. Bovik, Shape from texture by Wavelet-Based Measurement of Local Spectral Moments, IEEE 1992.

[30] M. Ortega, K. Chakrabarti, Sharad Mehrotra, T. S. Huang, Supported Ranked Boolean Similarity Queries in MARS, IEEE Transactions on Knowledge and Data Engineering, vol. 10, 6, pp. 905-924, 1998

[31] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechis, Image coding using wavelet transform, IEEE Trans. On Image Processing, vol. 1, n. 2, 1992, pp. 205-219.

INDICE DEI NOMI

INDICE

Finito di stampare
nel mese di ottobre 2001
dalla CUEN srl - Napoli
con DocuTech

Workshop General Chairs:
Prof. R. Chellappa *(Univ. of Maryland)*
Prof. L. Sansone *(Univ. di Napoli "Federico II")*

Workshop Program Chairs:
Prof. S. Adali *(Rensselaer Polytechnic)*
Prof. S. Tripathi *(Univ. of California Riverside)*

Steering Committee:
Prof. V. S. Subrahmanian *(Univ. of Maryland)*
Prof. S. Tripathi *(Univ. of California Riverside)*
Dr. D. Hislop *(US Army Research Office)*

Organizing Committee:
Prof. A. Chianese *(Univ. di Napoli "Federico II")*
Ing. A. Picariello *(Univ. di Napoli "Federico II")*
Dr. G. Boccignone *(Univ. di Salerno)*

mis '01 Capri

Segreteria Scientifica
Dipartimento di Informatica e Sistemistica
Via Claudio, 21 - 80100 NAPOLI
Tel. +39 081 7683827-26
E-mail: angelo.chianese@unina.it
Sito Web: www.cs.rpi.edu/mis2001

Segreteria Organizzativa
effe erre congressi snc
Via Coroglio 156, 80124 Napoli
Tel. +39 081 2302417 Fax +39 081 2301044
E-mail: frcongr@cittadellascienza.it