Content-Based Lecture Access for Distance Learning on the Web

Guang-Ho Cha
Department of Multimedia Engineering
Tongmyong University of Information Technology
Pusan 608-711, South Korea
82-51-629-7269
ghcha@tmic.tit.ac.kr

Chin-Wan Chung
Department of Computer Science
Korea Advanced Institute of Science and Technology
Taejon 305-701, South Korea
82-42-869-3537
chungcw@islab.kaist.ac.kr

ABSTRACT

Education and training are expected to change dramatically due to the combined impact of the Internet, database, and multimedia technologies. However, the distance learning is often impeded by the lack of effective tools and system to manage and retrieve the lecture contents effectively. This paper introduces a new approach to realize the distance learning on the Web. The approach involves: (1) The XML (eXtensible Markup Language)-based semistructured model not only to represent lecture contents but also to exchange them on the Web; (2) The technique to build structural summaries, i.e., schemas, of XML lecture databases. The structural summaries are useful for browsing the database structure, formulating queries, building indexes, and enabling query optimization; (3) Index structures to speed up the search to find appropriate lecture contents. Finally, an overall system architecture for Web-based distance learning is described.

Keywords

Distance learning, lecture modeling, lecture browsing, lecture querying, lecture indexing, semistructured data, XML, Web

1. INTRODUCTION

Education and training are expected to change dramatically due to the combined impact of the Internet, database, and multimedia technologies. Besides the economic impact – online education means less traveling, hence lower cost – the expectation is that the educational process itself will change radically [13]. Video is the most effective medium for providing remote and future students with a lecture because of its expressive power that combines images and voice. Moreover, the ability of recording and subsequently playing back live instruction sessions could significantly enhance the students' learning effectiveness because it allows them to review class lectures repeatedly. Unfortunately, however, the benefit of video-based lecture is often impeded by the fundamental difficulties with information retrieval: if one is trying to locate specific information on a video source, finding it can be a process that is time consuming and tedious. In addition, the contents of class lectures are diverse, and the same course can be given over and over again with different contents and structures by different instructors. Thus, we cannot conform the lecture content to a rigid, predefined schema. Three crucial issues that need to be addressed are: (1) the representation of lecture contents in a form that facilitates retrieval and interaction; (2) the structural summary of a lecture database that guides users to browse and query the database; and (3) the indexing scheme to expedite the search.

Browsing and *querying* in a lecture database for distance learning should provide the same ease of use as flipping through the pages of a book and scanning the table-of-contents and index pages to get ideas of the content quickly, and then gradually focusing on particular chapters or sections of interest. For a lecture database, this is not as straightforward as browsing and querying in a book. We have to identify the chapters, sections, and subsections of a lecture, and create table-of-contents and index pages for lecture, both structured and unstructured, so that we can get an overview and know where to find relevant contents.

The transformation of a simple lecture into a valuable educational tool requires five steps. First, we partition a lecture into individual lecture segments by exploiting the hierarchical structure of the lecture or book. A *lecture segment* consists of a set of lecture notes and any contiguous portion of a video *clip* which constitutes a digital video lecture. Each lecture segment is associated with the system-wide unique *identifier*. Second, we abstract the contents of lecture segments with text descriptions, meaningful attributes, and key images, and organize them into an effective structure that facilitates retrieval and interaction. Third, we need tools to aid the user for browsing a lecture database and formulating queries. Although it may be possible to manually browse a small database, in general forming a meaningful query is difficult without knowledge of the database structure. Fourth, we index all useful objects appearing in lecture segments to efficiently locate specific lecture segments of interest. Finally, we need query optimization techniques to reduce the search space and expedite the search since there are numerous query plans for each query.

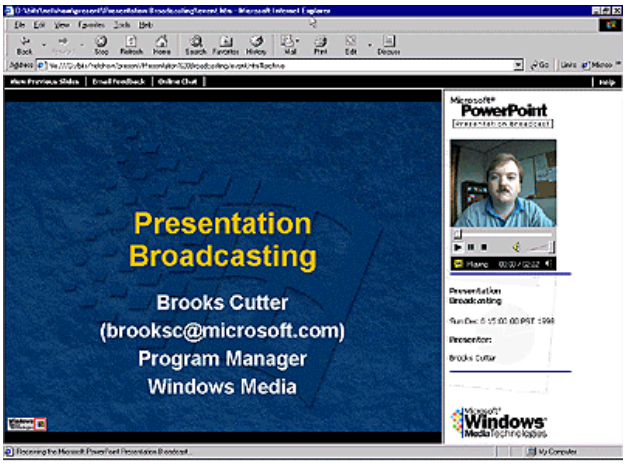


Figure 1. Online presentation window

In this paper, the XML-based semistructured model is introduced for content-based lecture access. It fully supports XML data and represents lecture contents without rigid, fixed schema. Database structure summarization, i.e., schema extraction, technique for irregular lecture database is used to guide users in database browsing and querying. Various index structures are presented to efficiently locate not only a specific lecture segment but also a collection of semantically related lecture segments. The system architecture for implementing the Web-based distance learning system is presented.

2. VIDEO SEGMENTATION

While a video clip consists of a sequence of frames, it is not meaningful to use the individual frames as the units for video retrieval. Rather, it is advantageous to identify meaningful segments of video to serve as retrieval units. As defined in [12], the fundamental unit of video production is a *shot* that consists of a contiguous sequence of video frames. While the video segmentation based on image processing techniques automates the process of video parsing, it has the following problems for distance learning:

- For a video clip of a class lecture, there can be no clear visual cue for shot change detection. Therefore, video segmentation using shot change detection algorithms would be difficult.
- Shots do not capture the underlying semantic structure of a class lecture, based on which the user may wish to browse and retrieve the video lecture.

On the ground of above motives, we do not pay special attention to the problem of the video segmentation based on image processing. Rather, we automatically extract descriptive text information from the instructor's lecture notes, and manually describe the necessary semantic video units and their contextual information. After that, the video lectures are automatically indexed, converted to a Web-ready format, and made available to end users through the Internet.

A lecture is organized into *presentation slides* (i.e., lecture notes) and video segments. Each slide corresponds to a single page course note assumed to be written in XML. Instructors lecture by showing electronic course slides, and recording of lectures is expected to capture the video of live lecture sessions. In our work, we define a *video shot* as the video segment synchronized with a single slide. The synchronization of slides with video segments can be easily made because instructors are required to explicitly switch slides during live lecture session. When students access a particular lecture in a course, they see the presentation similar to Figure 1. By allowing remote or future users to not just view presentation slides but also to see and hear the presenter, the instructor achieves a broader reach and increased productivity and the audience gets a richer experience that enables them to retain more information and saves on travel costs. However, the more important things we need for is to locate and retrieve a particular piece of the video lecture because watching the whole video is time consuming.

3. DATA MODEL

Data modeling deals with the problem of how to represent the data to facilitate users' access. To the best of our knowledge, there have been no efforts to model the lecture database. Previous work on data models for video data can be found in [3, 4, 5, 10, 12, 16, 17, 23, 24, 27]. Most of early research effort has been devoted to the shot-based video segmentation and each video shot is described using text descriptions and cinematic attributes. While this approach automates the video parsing, it lacks the contextual information between video shots, and thus it is difficult to describe the structure of the video content. Moreover, it lacks the flexibility and scalability since the video clips are segmented into independent shots. These problems are tackled by employing additional constructs to model contextual information. In [5, 10, 23, 27], a higher-level construct called *scene* is used to group together those shots that share some common properties. However, the two-layered scene-shot structure is not sufficient to represent the rich content of a class lecture. In [12, 24], instead of representing video information as independent shots, it divides the video sequence into a set of layers or *strata*. A stratum consists of the descriptive information such as title, keywords, frame delimiters, etc. The strata can be overlapped and nested. Though the *stratification* is more general than the scene-shot structure, it is also not sufficient to represent the class lecture by using a predefined set of attributes. The approach of [4] using the hierarchical Petri-nets focuses on only the spatio-temporal specification of events in video, and thus it is not concerned with the content of the class lecture. A data model called *VideoText* [17] employs free text annotation

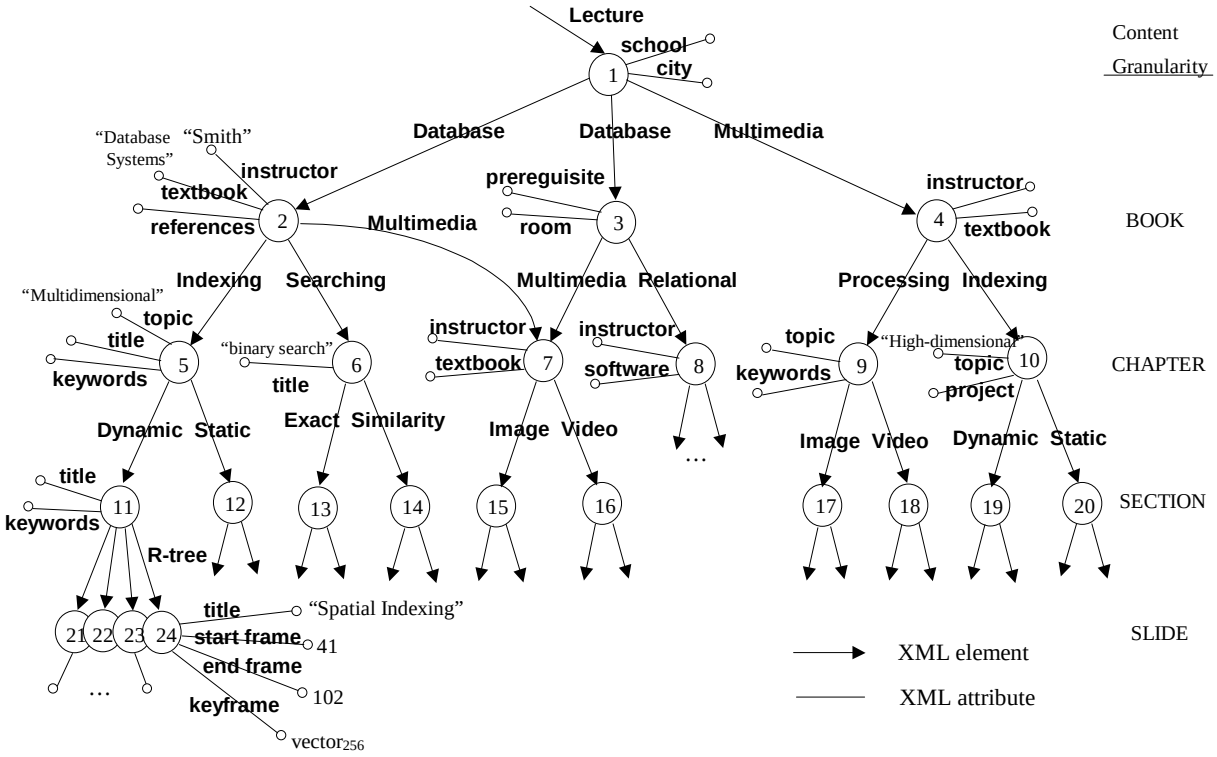


Figure 2. An example class lecture video database (Some nodes are omitted and only a few values of attributes are shown in the diagram)

on logical video segments rather than a fixed set of attributes. However, it lacks the contextual and structural relationships among the video annotations.

To tackle the problems of existing annotation-based video data models, we adopt the semistructured data model [1, 6, 22], specifically, XML [26]-based semistructured model. The motivation to employ the semistructured model comes from the need to provide the lecture content description with flexibility and diversity. Because the lecture contents are diverse and rich, we cannot conform the lecture database to a rigid, predefined schema. Moreover, the motivation to fully support XML data is to exchange lecture data on the Internet. By semistructured data we mean data that has no absolute schema fixed in advance, and whose structure may be irregular or incomplete. Like in the standard model [1, 6, 22] for semistructured data, a lecture database is thought of as a labeled directed graph. For example, Figure 2 depicts a portion of a lecture database containing three class lectures (two for a database course and one for a multimedia course). Each node corresponds to an *XML element* and can have *attributes* depicted as small circles in Figure 2. Our example database is almost tree-structured because of the hierarchical nature of the book for lecture even though the semistructured model permits arbitrary graph-structured databases. Each level of our example lecture database represents the level of content granularity. For example, we can assume that nodes 2 through 4 are in the level of book, nodes 5 through 10 are in the level of chapter, nodes 11 through 20 are in the level of section, and so on.

Unlike the standard semistructured model, our data model fully supports the XML data. In other words, it allows us to associate attributes with graph nodes (XML elements). In our data model we call the nodes *lecture objects* (LOs) in which the video segments and presentation slides for lecture are associated. An LO can be viewed as a 6-tuple (PID, OID, a set of video segments, a set of presentation slides, a set of sub-elements, a set of attributes). We should note that the elements and the attributes attached to LO are not pre-fixed. Each LO has a unique *object identifier* (OID), such as 1 to 24 in Figure 3, and outgoing edges that correspond to its sub-elements. Every LO belongs to a certain type and the type is identified by a *path identifier* (PID). In our model, a type is defined by a path on the extracted schema graph, which will be described in the next section. Labels are attached to the edges and they serve as *names* for LOs or attributes. Our example database in Figure 2 contains one root LO which represents the Lecture database and contains three sub-LOs, two Databases and one Multimedia. Database LO 2 has three attribute-value pairs

describing its instructor, textbook, and references, while Database LO 3 has two attributes prerequisite and room. Unlike the standard semistructured model, sub-LOs under an LO in our model are ordered to reflect the timing sequence of the video segments associated with them. We can see that the database structure based on the semistructured model is irregular since, for example, two Database objects (LO 2 and LO 3) have different structures.

4. SUMMARIZING LECTURE DATABASE

Two completely different types of lecture retrieval requests can be expected from the end-user:

- *Querying*: The user retrieves particular lecture objects for viewing or reuse.
- *Browsing*: The user traverses a lecture database along the semantic links.

A query processor should respond to both types of retrieval requests by providing the user with query formulation tools for querying and optimal starting points for browsing. When we model a lecture retrieval request as an iterated sequence of querying and browsing, each step should act as an information *filter* reducing the search space and give a more refined search space to the next step. In a small database, although it may be possible to browse the whole database, in general it is difficult and tedious to browse a large database. It is reasonable to pose a query at the start by using some attributes. However, since our lecture database is based on the XML-based semistructured model, i.e., it is schemaless, it needs a tool that assists users in query formulation by providing the information (i.e., schema) summarizing the database structure. The schema allows users to browse and query easily through the database. Also, it improves the system performance greatly by enabling to take advantage of indexes and query optimization.

Figure 3 shows the structural summary of the lecture database given in Figure 2. A rectangle corresponds to an XML element in the database, and small black circles denote XML attributes in the XML element. Every XML element of an original database is described exactly once in the structural summary, regardless of the number of times it appears in that database. There is no XML element that does not appear in the original database. From the structural summary, a user can interactively query and browse the graph-based database. Clicking on a rectangle on the structural summary expands or collapses LOs. The white rectangle indicates that the LO has been expanded and the black rectangle indicates that the LO has not. For example, Database and Multimedia LOs have been expanded, while Dynamic and Static LOs have not.

We develop a summarizer that extracts a schema from an irregular lecture database. *DataGuids* [14] are concise and accurate summaries of semistructured databases. Unfortunately, however, DataGuides can be very expensive to compute since they require a powerset construct over the underlying database. For a general graph, the algorithm to construct a DataGuide can be exponential in space and time with respect to the size of the underlying database. Buneman et al. [7] constructs database summaries based on the computation of *simulations* or *bisimulations* [15] for which efficient construction algorithm exists. The size of a database summary based on the simulation is guaranteed to be

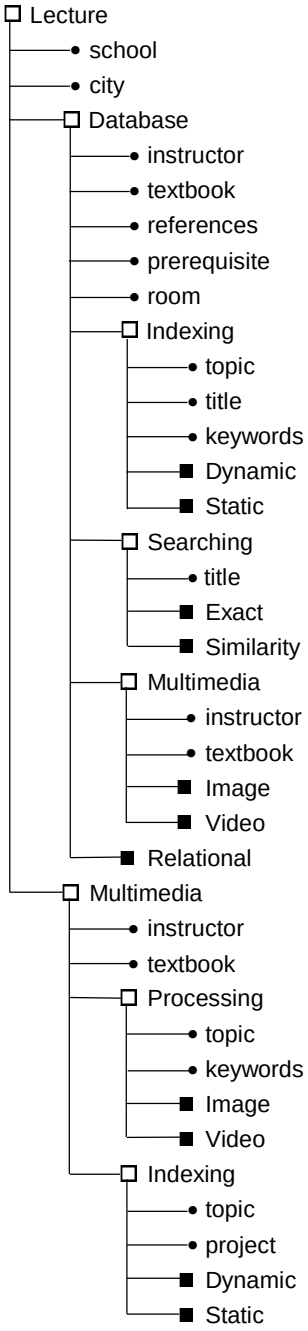


Figure 3. Structural summary of the example database in Figure 2

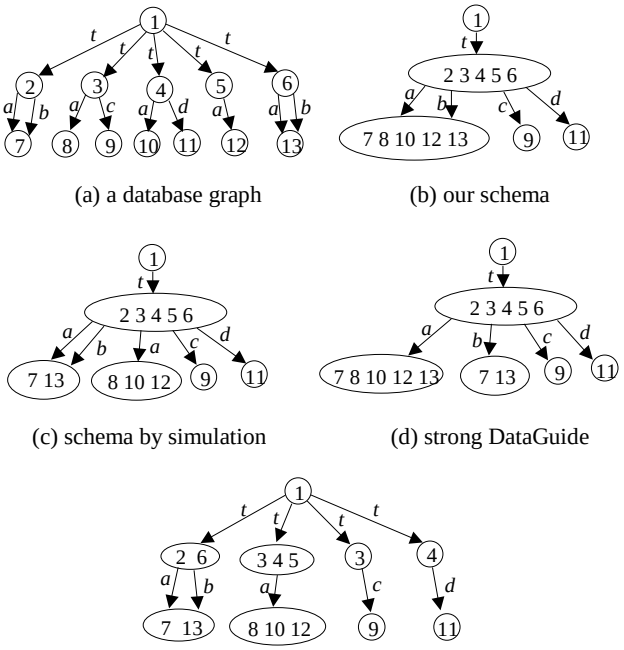


Figure 4. Comparison of database schemas

at most linear in the underlying database size. However, they still have redundancy in edges of the schema graph. Nestorov et al. [20] developed a technique to extract schema based on the greatest fixpoint semantics of monadic datalog program and the clustering. Although their method can reduce the size of the schema to a desired size, its running cost may be excessive because of its complexity of algorithm. Moreover, since it has to perform clustering to get a desired schema size, it may be difficult to use this method in dynamic environments.

We develop a new technique to improve the running time to build the structural summary of a database. In our method, the structural summary does not have any redundancy in nodes and edges in a schema graph. We can best explain the difference among our technique, Buneman et al.'s, DataGuide, and Nestorov et al.'s. Figure 4(a) illustrates a database graph *DB* and 4(b) is our summary of *DB*. Figures 4(c), 4(d), 4(e) show Buneman et al.'s schema based on simulation, strong DataGuide, and Nestorov et al.'s minimal perfect typing, respectively. We compare the schemas by their size. DataGuides require a powerset construct over the underlying database, which in the worst case can be of exponential cost. As you can see in Figure 4(d), elements 7 and 13 are replicated in nodes in the DataGuide. The schema based on simulation guarantees its size, that is, the size of the schema is at most linear in that of the database. However, as we can see in Figure 4(c), edge *a* outgoing from the same node is replicated. Figure 4(e) also shows redundancy in nodes and edges. On the other hand, our database schema in Figure 4(b) does not have any redundancy in nodes and edges. The compactness of our schema results in efficiency in query evaluation as well as in database summarization.

4.1 The Algorithm

We define some terminologies before proceeding.

Definition 1. A *data object* is a node, i.e., LO, in a database graph.

Definition 2. A *target set* for a path *l* is a set of data objects that can be reached by traversing a path *l* in a database graph.

Definition 3. A *schema object* is a node in the schema graph that corresponds to a target set of a path *l* in a database graph.

The schema extraction is easy to implement with our algorithm. The root data object becomes a root schema object. In a depth-first fashion, we extract all child schema objects reachable by all unique paths outgoing from a schema object. Each time we encounter a new target set for a unique path *l*, we create a new schema object *s*. If we reach a schema object *s* via a path *l* and a data object *o* is already included in the schema object *s* with a different path *m*, rather than creating a new schema object we instead add an edge *l* to the schema object *s*. The algorithm is specified as follows.

Algorithm ExtractSchema(*o*)
// Input: root oid *o* of a database
// Output: database schema *s*
{
  s := CreateSchemaObject();
  Insert {*o*} to *s*;
  RecursiveMake(*s*);
}

Algorithm RecursiveMake(*s*)
{
  Let *S* be a set of current target sets under *s*;
  Let S_j denote a certain target set included in *S*;
  For each unique label l_i outgoing from *s* {
    o := target set reachable by l_i;
    If (*o* and S_j have data objects in common) {
      Add an edge l_i from *s* to the schema object corresponding to S_j;
      $S_j := o \cup S_j$;
    }
    Else {
      s_2:= CreateSchemaObject();
      Insert s_2 to *s*;
      Add an edge l_i from *s* to s_2;
      RecursiveMake(s_2);
    }
  }
}

5. INDEXING

In traditional databases, an index is created on an attribute in order to locate objects for particular attributes quickly. Despite the cost of maintenance and the added storage, indexes are useful and integral part of all database systems. In lecture databases with XML-based semistructured data model, such a *value index* alone used in traditional databases is not sufficient since we have to efficiently traverse the database graph. We need several index structures that are useful for finding relevant objects, specific edges, and paths within the database. Since the access to the lecture databases tends to be read-intensive, maintaining extensive index structures to speed up query processing is justified.

In this paper, we propose two new index structures called *P-index* (*path index*) and *LPC-index* (*local polar coordinate index*) to index paths on the database graph and images in the lecture video, respectively. Example queries used in this paper are formulated in the query language similar to the *Lorel* query language [2] which is an extension of OQL [8]. The example queries are executed over the example lecture database in Figure 2.

5.1 P-index

Here we introduce the P-index to index the path on the database graph with some motivating examples.

5.1.1 Motivating Examples

Example 1: Retrieve lecture objects whose title is "Spatial Indexing."

Select x
Where ∗.x.title = "Spatial Indexing"

The wildcard "*" means any path of length 0 or more. For this type of value queries, we can consider the B^+-tree [11] index structure for the attribute *title* and can get the result of LO 24. However, if the structural and contextual relationships between lecture objects are not provided together with the target lecture object, potentially useful results may not be found only by a simple value-matching search. For example, the lecturer may assume that the reader would not read the lecture object unless he had read earlier lecture objects. As a result, the reader may not understand the context if he does not read earlier lecture objects. Thus it is desirable to output the query result as follows so that the reader can traverse the graph hierarchy if he wants:

- Lecture (LO 1)
  - Database (LO 2)
    - Indexing (LO 5)
      - Dynamic (LO 11)
        - R-tree (LO 24)

Example 2: Retrieve Database lecture objects in which the title is "Spatial Indexing."

Select x
From Database x
Where x.∗.title = "Spatial Indexing"

This type of queries could be very expensive without appropriate indexes because we have to traverse backward to every Database object after identifying lecture objects whose title is "Spatial Indexing." We can also employ a top-down execution strategy in which we find all Database lecture objects and begin at them and evaluate every path in a forward manner to check if their descendants' title is "Spatial Indexing." It should be obvious that this query execution is also costly since we have to traverse every path from Database objects. To support this type of queries, we provide the P-index that stores all objects along the backward path from the qualifying objects to the root of the database graph.

5.1.2 P-Index Structure

When we consider the indexing of graph paths, it should be obvious that data models with a more relaxed typing paradigm have to impose user-specified and dynamically controlled type constraints on attributes and/or paths that are indexed. We impose types on the labeled paths on the structural summary graph of the database. For example, considering the path Database.Indexing.Dynamic.R-tree, four types are imposed: Database, Database.Indexing, Database.Indexing.Dynamic, and Database.Indexing.Dynamic.R-tree. Each type (or path) is uniquely identified by its *path identifier* (PID).

The structure of the P-index is based on the B^+-tree. The P-index consists of internal and leaf nodes as in other dynamic index structures. The internal node of the P-index has the same structure as that of the B^+-tree. The leaf node has a format different from that of an internal node. It consists of f index entries and each index entry has a form shown in Figure 5, where f is the fanout of a leaf node. For path indexing, the P-tree maintains in a leaf node the lecture objects on the label paths from the qualifying objects to the root. If the size of a leaf node entry exceeds a page size, additional overflow pages are allocated. The P-index is

key value	overflow page pointer	no. of PIDs	PID_1	no. of OIDs	$\{OID_{11},\ldots, OID_{1i}\}$	…	PID_n	no. of OIDs	$\{OID_{n1},\ldots, OID_{nk}\}$

Figure 5. An entry of a leaf node of the P-index

somewhat similar to the class-hierarchy indexing [18] used in object-oriented databases. The class-hierarchy indexing maintains one index on a common attribute for a hierarchy of classes. On the other hand, there is no concept of a common attribute in the irregular semistructured database. Instead, the P-index maintains one index on every path from the qualifying objects to the root.

5.2 LPC-index

Users can query a database to find lecture objects whose video segments include an image similar to a user-specified image.

Example 3: Retrieve lecture objects including a video segment whose key frame is similar to a given image *P*.

Select x
Where x.*.keyframe similarto *P*

Image query is performed over objects that have key frame image. The main issue in the image indexing is the *curse of dimensionality* [21], i.e., the search performance drastically deteriorates if the dimensionality (i.e., the number of feature elements of an image) goes high (e.g., larger than 10). In high dimensional data space, the performance of conventional index structures degenerates to being worse than that of the brute-force linear scan that compares the query object to each object sequentially. We propose the *local polar coordinate index* (LPC-index) for indexing images with high-dimensional feature vector. The performance of the LPC-index has been verified in high-dimensional data space (e.g., > 100).

The LPC-index employs a *filter-based* approach in which feature vectors are represented as compact approximations to the original vectors and by first scanning these smaller approximations, only a small fraction of the vectors are visited. The basic idea of the LPC-index is as follows: First, the LPC-index assigns the same number of bits *b* to each dimension of the feature vector and divides the whole data space into 2^{bd} cells, where *d* is the number of dimensions. Second, the LPC-index approximates the vector $\boldsymbol{p}$ using the polar coordinates (r, θ) within the cell in which $\boldsymbol{p}$ lies. As illustrated in Figure 5, the local origin *O* of each cell is determined by the lower left corner of the cell. The radius *r* is computed by the distance between the local origin *O* and the vector $\boldsymbol{p}$. The angle θ is computed by the angle between the vector $\boldsymbol{p}$ and the diagonal from the local origin to the opposite corner. As a result of this approximation, the vector $\boldsymbol{p}$ is represented by the triplet $\boldsymbol{a} = < c, r, \theta >$, where c, r, θ denote the approximation cell, the radius, and the angle between $\boldsymbol{p}$ and the main diagonal, respectively. The complete LPC-index is an array of approximations for all vectors. In spite of taking a small amount of information to represent the local polar coordinates, this information significantly enhances the discriminatory power of the approximation in high dimensions.

When searching for the nearest neighbor image, the entire approximation file is scanned and lower bound (d_{min}) and upper bound (d_{max}) on the distance from the image vector $\boldsymbol{p}$ to the query vector $\boldsymbol{q}$ are determined such that

$$d_{min} \le L_2(\boldsymbol{p}, \boldsymbol{q}) \le d_{max}$$

where L_2 is the Euclidean distance. The d_{min} and d_{max} are computed as follows:

$$d_{min}^2 = |\boldsymbol{p}|^2 + |\boldsymbol{q}|^2 - 2|\boldsymbol{p}||\boldsymbol{q}|\cos|\theta_1 - \theta_2|$$
$$d_{max}^2 = |\boldsymbol{p}|^2 + |\boldsymbol{q}|^2 - 2|\boldsymbol{p}||\boldsymbol{q}|\cos(\theta_1 + \theta_2)$$

where θ_1 is the angle between the vector $\boldsymbol{p}$ and the diagonal of the cell in which $\boldsymbol{p}$ lies and θ_2 is the angle between the vector $\boldsymbol{q}$ and the diagonal of the cell in which $\boldsymbol{p}$ lies. Assume δ is the smallest upper bound found so far. If an approximation is encountered such that its lower bound exceeds δ, the corresponding object can be eliminated since at least one better candidate exists. Analogously, we can define a filtering step when the *k* nearest neighbors must be retrieved. After the filtering step, a small set of candidates remain. These candidates are then visited in increasing order of their lower bound on the distance to the query object $\boldsymbol{q}$, and the accurate distance to $\boldsymbol{q}$ is determined. However, not all candidates must be accessed. Rather, if a lower bound is encountered that exceeds the *k*-th nearest distance seen so far, the LPC-index stops.

Figure 7 shows the result of elapsed time experiments of the 10-nearest neighbor (NN) search on 1,000,000 objects for the linear can, the VA-file [25], and the LPC-index. The *Scan* algorithm is a simple linear scan of the vectors themselves, maintaining a ranked list of the 10 NN vectors encountered so far. The VA-file is the only NN index structure that results in exact results and outperforms the linear scan. In the synthetic (random and skewed) data set, the LPC-index outperforms the VA-file and the Scan by a factor of 2 and 3 on the average, respectively. In real data set, the performance of the VA-file decreases drastically as soon as its vector selectivity falls below a certain threshold point. In a 256-dimensional real image set, the performance of the VA-file using 8 bits per dimension for a cell degenerates close to that of the Scan. Even in this worst case of data distributions, the LPC-index shows a performance improvement over the Scan, reflecting the reduction of data as much as possible by approximation.

6. OVERALL SYSTEM ARCHITECTURE

We are currently developing a Web-based information system for distance learning called COVA (COntent-based Video Access) within our CyberUniversity project. The initial system was entirely written in Java language. The user can access the lecture database from anywhere using a popular Web browser. The system includes seven major

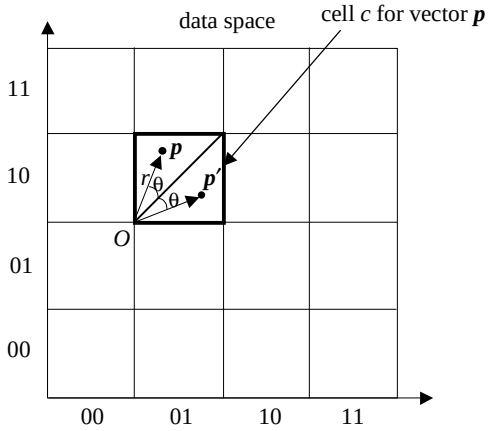
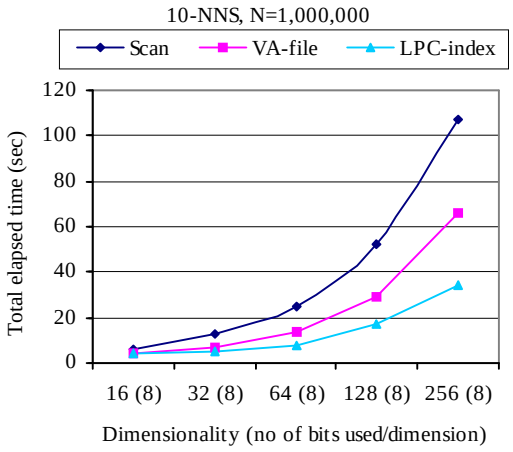
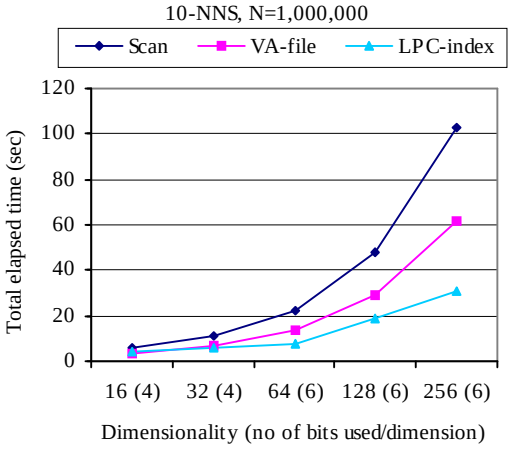


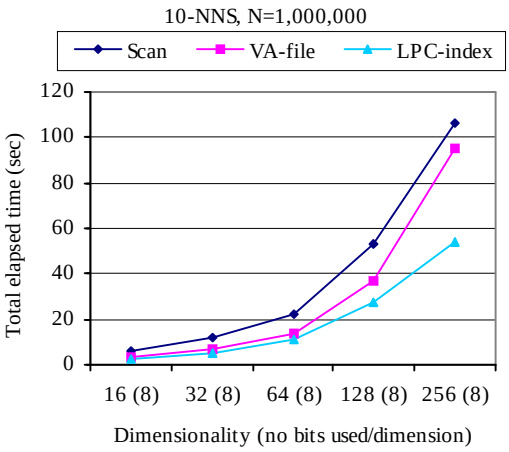
Figure 6. 2-dimensional vector ***p*** and its approximation (c, r, θ) in the LPC-index



(a) Random distribution



(b) Skewed (Zipf) distribution



(c) Real image data

Figure 7. Total elapsed time

components for text processing and annotation, video processing and annotation, structural summarization, indexing, storage management, browse/query processing, and streaming media delivery (see Figure 8).

Text Processing and Annotation

The text processing automatically extracts the titles, free text, and keywords from the instructor's presentation slides and attaches them to each LO. Using the text annotator, we can attach additional descriptive attributes to each VO, and link and group related LOs together so that the class lecture has contextual information.

Video Processing and Annotation

The video processing and annotation detects meaningful units of video and charaterizes these units. Although we do not focus on this issue in this paper, in fact, our system is ready to incorporate existing image and video processing techniques to support visual video querying and browsing.

Structural Summarization

The structural summarizer builds a schema of a semistructured database to provide the benefits of the schema. Users exploit the structural summaries for browsing database structure and formulating queries. The indexing scheme and query processor of COVA rely on the structural summaries to build indexes and to devise efficient plans for computing query results, respectively.

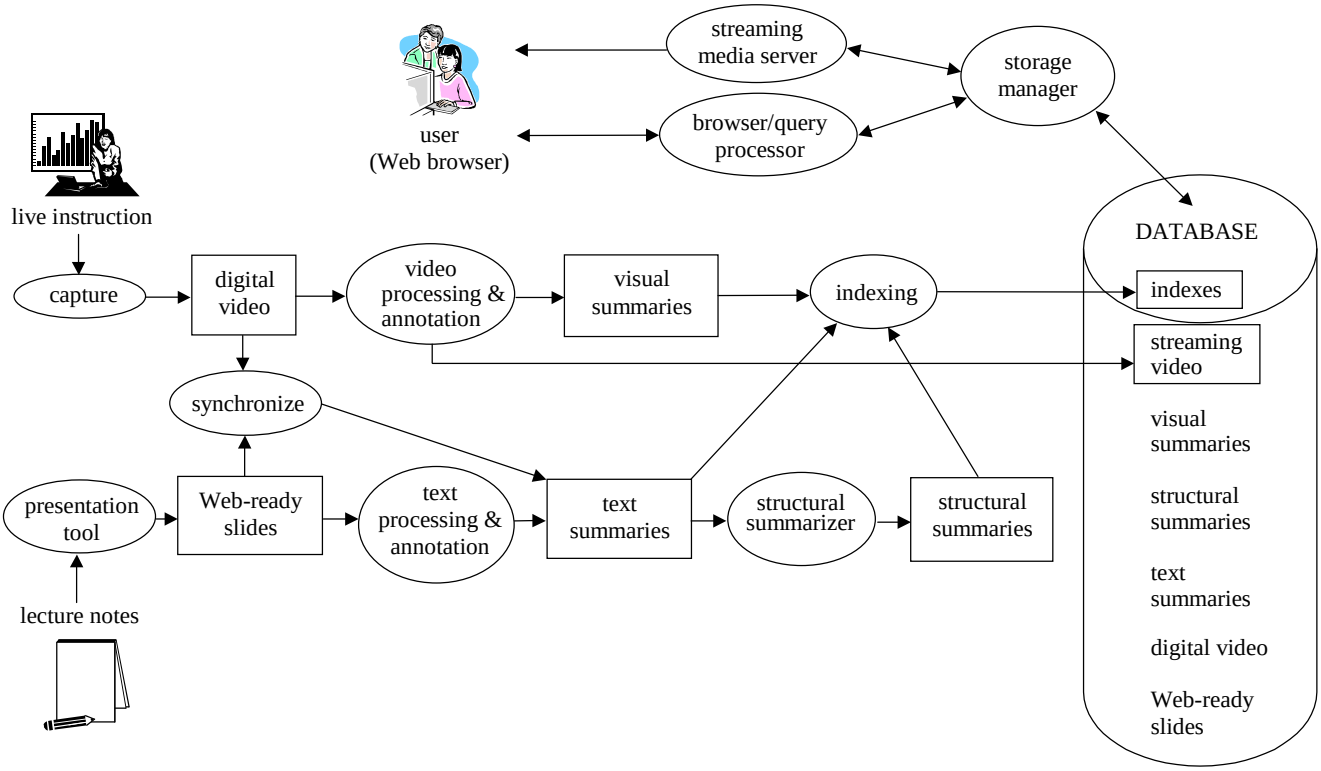


Figure 8. System architecture

Indexing

Since the charateristics of database applications such as distance learning and training tend to be read-intensive, the extensive use of index structures to speed up query processsing is justified. COVA currently employs four index structures: B^+-tree, inverted-file index, LPC-index, and P-index.

Query Processing and Browsing

In graph-based data model, there are many ways to execute a single query. The optimal query plan depends not only on the values in the database but also on the shape of the graph containing the data. Three types of query execution strategies are general: top-down, bottom-up, and hybrid strategies. The *top-down* strategy begins at the top object and evaluates the From clause by processing each simple path expression in a forward manner. This strategy results in a depth-first traversal of the graph following edges that appear in the path expressions. The *bottom-up* strategy first identifies all objects that satisfy the Where clause. Once we have an object satisfying the predicate, we traverse backwards through the data, going from child to parent, matching in reverse the path expressions appearing in the Where and then in the From. The *hybrid* strategy operates both top-down and bottom-up, meeting in the middle of a path expression. By intersecting the sets of objects resulted from both strategies we find the result of the query.

One important thing in our browser/query processor is the user interface that integrates the navigational object browsing and declarative querying. Web users are familiar with specifying a simple query to begin a search and further exploring and refining the results. In other words, it represents querying as an extension of browsing.

Storage Manager

The storage manager is concerned with the allocation and clustering of data objects and indexes on disk, and the movement of data between disk and main memory. One of the major issues is how to incorporate the semantics of the semistructured model in the storage manager. In most graph-based data model, objects are identified by their incoming labels. This basic assumption is used by the storage manager, which clusters a database by grouping together objects with identical incoming labels on disk. COVA also employs the *segmented-page indexing* (*SP-indexing*) scheme [9] for clustering of indexes. The SP-indexing scheme uses two kinds of I/O unit: *page* for random disk accesses and *segment* for sequential disk accesses. The SP-indexing avoids that the related index nodes are scattered widely on the disk by storing them contiguously within a segment. It also provides a compromise between optimal

index node clustering and excessive full index reorganization overhead.

Streaming Media Server

The streaming media server is responsible for delivering the video at the exact data rate associated with the compressed audio and video streams, and it responds to the feedback from the client.

7. CONCLUSIONS

The wide spread adoption of Internet streaming video and the advances of multimedia and database technologies present a new opportunity of education and training. We presented a new approach to the distance learning based on the XML-based semistructured model. By employing this model, we could provide the lecture contents with flexibility and diversity as well as exchange them conveniently on the Internet. Based on this model, we described the technique to extract schemas from a graph-based database. In irregular semistructured database, without schema, it is difficult to query and browse the database, to construct indexes, and to perform query optimization. Two index structures for path queries and image queries were also introduced to speed up the search. Read-intensive lecture database applications justify the extensive use of index structures to speed up the query processsing. Finally, we presented the overall system architecture for implementing the video-based distance learning system. We believe that our system will provide a valuable education and training tool for remote or future users.

REFERENCES

[1] S. Abiteboul, "Querying Semistructured Data," *Proc. of the International Conference on Database Theory*, pp. 1-18, 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, "The Lorel query language for semistructured data," *International Journal on Digital Libraries*, Vol. 1, No. 1, pp. 68-88, 1997.

[3] P. Aigrain, HongJiang Zhang, and D. Petkovic, "Content-based representation and retrieval of visual media: A state-of-the-art review," *Multimedia Tools and Applications*, Vol. 3, No. 3, pp. 179-202, 1996.

[4] W. Al-Khatib and A. Ghafoor, "An Approach for Video Meta-Data Modeling and Query Processing," *Proc. of the ACM International Multimedia Conference*, 1999.

[5] R.M. Bolle, B.-L. Yeo, and M.M. Yeung, "Video Query: Research Directions," *IBM Journal of Research and Development*," Vol. 42, No. 2, pp. 233-252, 1998.

[6] P. Buneman, "Semistructured Data," *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 117-121. 1997.

[7] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu, "Adding Structure to Unstructured Data," *Proc. of the International Conference on Database Theory*, 1997.

[8] R.G.G. Cattell, *The Object Database Standard: ODMG-93*, Morgan Kaufmann, San Francisco, CA, 1994.

[9] G.-H. Cha, X, Zhu, D. Petkovic, and C.-W. Chung, "Segmented Page Indexing for Range Queries in Multidimensional Data Spaces," *IBM Research Report* RJ 10176 (95050), 1999.

[10] T.-S. Chua and L.-Q. Ruan, "A Video Retrieval and Sequencing System," *ACM Transactions on Information Systems*, Vol. 13, No. 4, pp. 373-407, 1995.

[11] D. Comer, "The Ubiquitous B-tree," *ACM Computing Surveys*, Vol. 11, No. 2, pp. 121-137, 1979.

[12] G. Davenport, T.A. Smith, and N. Pincever, "Cinematic Primitives for Multimedia," *IEEE Computer Graphics and Applications*, pp. 67-74, 1991.

[13] A. Ginsberg, P. Hodge, T. Lindstrom, B. Sampieri, and D. Shiau, "The Little Web Schoolhouse:" Using Virtual Rooms to Create a Multimedia Distance Learning Environment," *Proc. of the ACM International Multimedia Conference*, pp. 89-98, 1998.

[14] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," *Proc. of the International Conference on Very Large Data Bases*, pp. 436-445, 1997.

[15] M. Henzinger, T. Henzinger, and P. Kopke, "Computing simulations on finite and infinite graphs," *Proc. of Symp. on Foundations of Computer Sciences*, pp. 453-462, 1995

[16] R. Hjelsvold and R. Midtstraum, "Modeling and Querying Video Data," *Proc. of the International Conference on Very Large Data Bases*, pp. 686-694, 1994.

[17] H. Jiang, D. Montesi, and A.K. Elmagarmid, "Integrated Video and Text for Content-Based Access to Video Databases, *Multimedia Tools and Applications*, Vol. 9, pp. 227-249, 1999.

[18] W. Kim, K.-C. Kim, and A. Dale, "Indexing Techniques for Object-Oriented Databases," Object-Oriented Concepts, Databases, and Applications, pp. 372-394, Eds. W. Kim and F.H. Lochovsky, ACM Press, 1989.

[19] J. McHugh and J. Widom, "Query Optimization for XML," *Proc. of the International Conference on Very Large Data Bases*, pp. 315-326. 1999.

[20] S. Nestorov, S. Abiteboul, R. Motwani, "Extracting Schema from Semistructured Data," *Proc. of ACM SIGMOD*, pp. 295-306, 1998.

[21] W. Niblack et al., "The QBIC Project: Querying Images By Content Using Color, Texture, and Shape," *Proc. of the SPIE Conf. on Storage and Retrieval for Image and Video Databases II*, pp.173-187, 1993.

[22] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange Across Heterogeneous Information

Sources," *Proc. of the International Conference on Data Engineering*, pp. 251-260. 1995.

[23] Y. Rui, T.S. Huang, and S. Mehrotra, "Exploring Video Structure Beyond the Shots," *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, pp. 237–240, 1998.

[24] T.G.A. Smith and G. Davenport, "The stratification system: A design environment for random access video," *Proc. of the Workshop on Networking and Operating System Support for Digital Audio and Video*, 1992.

[25] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. of the 24th VLDB Conference*, pp. 194-205, 1998.

[26] The World Wide Web Consortium (W3C)'s XML web page, 1998. http://www.w3c.org/XML/.

[27] B.-L. Yeo and M.M. Yeung, "Retrieving and Visualizing Video," *Communications of the ACM*, Vol. 40, No. 12, pp. 43- 52, 1997.

[28] M. Zloof, "Query By Example," *IBM Systems Journal*, Vol. 16, No, 4, pp. 324-343, 1977.