비디오 문서의 구조 질의를 위한 효율적 인덱스 구조

이 용 규†

요 약

최근에 비디오 데이터베이스에 대하여 관심이 집중되고 있다. 비디오 문서도 텍스트 문서와 마찬가지로 계층적 논리 구조를 포함하고 있다. 사용자가 구조 질의로 이 구조를 활용한다면, 내용 질의에만 의존하는 것보다 더 큰 효과를 거둘 수가 있다. 구조 질의를 효율적으로 처리하기 위해서는 비디오의 문서 요소에 빨리 접근 할 수 있도록 지원하는 인덱스 구조가 필수적이다. 이 논문에서는 트리 구조의 비디오 문서 모형을 제시하고 비디오 문서를 위한 새로운 역 인덱스 구조를 제안한다. 그리고 이 인덱스 구조의 저장 장소 요구량과 디스크 접근 시간을 평가하고, 분석 결과를 제시한다.

An Efficient Index Structure Supporting Structure Queries for Video Documents

Yong Kyu Lee†

ABSTRACT

Recently, much attention has been focused on video databases. Video documents also have a hierarchical logical structure like text documents. By exploiting this structure using structure queries, users can obtain greater benefits than by using only content queries. In order to process structure queries efficiently, an index structure supporting fast video element access must be provided. However, there has been little attention to the index structure for video documents. In this paper, we present a tree-structured video document model and a new inverted index structure for video documents. We evaluate the storage requirement and the disk access time of the scheme and present the analytical results.

1. Introduction

Much attention has been focused on video databases including [13, 16, 18]. Video documents also have hierarchical structures as with text documents [12, 17]. The video elements, such as segments, events, and shots, compose a video document. By exploiting this structure using structure queries, users can obtain greater benefits than by using only content queries. Structure query languages which traverse the logical structure of text documents have been proposed in [4, 10]. However, there has been little attention to the structure query in video databases. In this paper, we propose solutions for some issues in video databases including video document model, structure query language, and index structure.

In order to support video structure queries, we have to maintain information about video structures. For this purpose, we use a

† 종신회원 : 동국대학교 컴퓨터정보통신공학부
논문접수 : 1997년 8월 8일, 심사완료 : 1998년 3월 2일

tree-structured video document model which forms a video tree with video elements for each video. Then, we assign a specially designed unique element identifier (UID) to each video element. By letting the UID carry information about document structure, the UID's of the ancestor or descendent nodes can be obtained directly from the UID. This scheme has advantage over those used for text document structure query processing in [1, 3] because we do not use other data structures such as parse trees and database relations.

Many index structures supporting fast data retrieval have been proposed for databases and information retrieval systems [2, 5, 8, 11, 15, 16]. We also need an efficient index structure for video databases. Video indexes can be built from video annotations and captions. In order to perform structure queries efficiently, an index structure which supports fast element access must be provided. For this purpose, index structures based on the conventional inverted index can be used. Sacks-Davis et al. [14] have proposed some possible inverted index structures for structure query processing. However, every scheme requires considerable storage overhead to store document element identifiers in the inverted list.

In this paper, we propose a new inverted indexing scheme which reduce the storage overhead considerably. We analyze storage requirements and disk access times of this scheme. Our approach can be applied to text databases as well.

2. Related Work

Oomoto and Tanaka [13] have presented a video-object database system named OVID. In their system, users define attribute structures for video objects without defining a specific database schema. By using the concept of interval inclusion, annotations of a video object can be inherited to its subobjects. In order to create new video objects, they have defined composition operations such as interval projection, merge, and overlap. A query language named VideoSQL has been used to retrieve video objects based on the content. However, they have not considered structure queries and an index structure supporting the interval inclusion hierarchy.

Weiss et al. [18] have defined an algebraic video data model. A video segment can be associated with descriptions extracted from text captions or images, and with users' annotations or structural and temporal information between video segments. The descriptions and annotations are used to support content based access to video. Even though they support the query based on the video structure, it is primitive compared to the structure queries of the text document system, and it can be supported only if users associate the structural information to the video segment. Moreover, they have not described an index structure supporting video segment access.

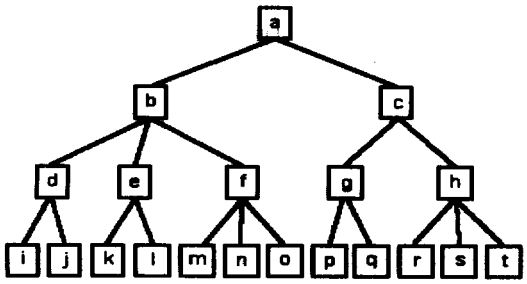
3. Video Document Model

Video documents have a hierarchical logical structure like text documents. The structure of video documents has been studied in [12, 13, 17]. The video elements, such as segments, events, and shots, compose a video, which has a hierarchical structure as shown in (Fig. 1)

(Fig. 1) Video Element Hierarchy

The video model, represented as a tree, forms a conceptual video structure on which a query language is based. Users are able to traverse the video tree through queries which exploit the structure.

An example of the video tree according to the video element hierarchy is illustrated in (Fig. 2)



(Fig. 2) Video Document Tree

We interpret a video structure as a k-ary tree [7] and assign each element a UID according to the level-order traversal order. For example, for the video tree of (Fig. 1), we assign UID's as 〈Table 1〉 assuming a 3-ary tree. Because we assume the tree is complete, there are some virtual nodes which do not exist.

〈Table 1〉 Element Identifiers for the (Fig. 2)

a	1	k	17
b	2	l	18
c	3	m	20
d	5	n	21
e	6	o	22
f	7	p	23
g	8	q	24
h	9	r	26
i	14	s	27
j	15	t	28

The UID of the parent or i-th child of a node whose UID is i can be obtained by the following functions.

$$parent(i) = \left\lfloor \frac{(i-2)}{k} + 1 \right\rfloor$$

$$child(i, j) = k(i - 1) + j + 1$$

These functions are used during structure query processing to evaluate the branch and list expression.

4. Video Structure Queries

Our query language[9] has a similar format to the SQL. In the query, users specify required video elements using two kinds of path expressions: branch expression and list expression. The branch expression is for moving forward and backward through the tree branches, and the list expression is for selecting an element from a list. The following are some examples.

○ Find the events which have a shot about "war."

```
find e
from v:video, e:v.segment.event
where exists(e.branch[1] contains "war")
```

○ Find the segments whose last event contains "farewell."

```
find e.branch[-1]
from v:video, e:v.segment.event()
where e contains "farewell"
```

○ Find the first shot of the events which contain "mountain."

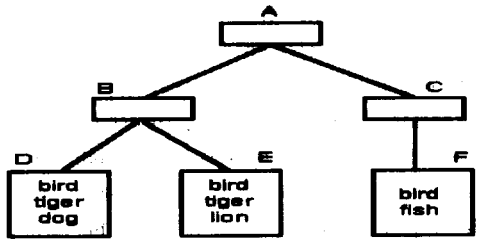
```
find e.shot[1]
from v:video, e:v.segment.event
where e contains "mountain"
```

In the above example, the branch expression branch[1] denotes the nodes reachable with one downward jump and branch[-1] means the parent node. The link expression event()]

represents the last event and shot[1] the first shot.

5. Video Index Structures

Since users want to access any kind of video element in the video tree, it is necessary to build an index structure supporting element access from video annotations or captions. Suppose that a video has three leaf nodes with keywords extracted from annotations as shown in (Fig. 3)



(Fig. 3) Video Tree with Keywords

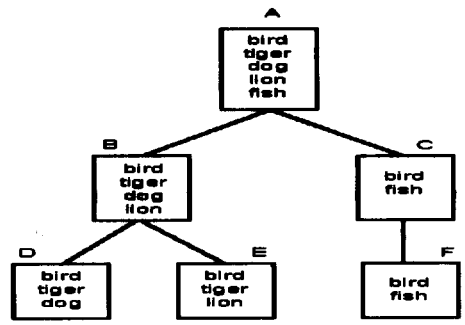
Even though an internal node has no associated data, the data of its subtrees should be considered as its data. Thus, the index for each node is as follows:

index(A) = {bird, tiger, dog, lion, fish}
index(D) = {bird, tiger, dog}
index(B) = {bird, tiger, dog, lion}
index(E) = {bird, tiger, lion}
index(C) = {bird, fish}
index(F) = {bird, fish}

5.1 Inverted Index with Replication(IR)

This approach replicates all keyword items of the children to their ancestor nodes as shown in (Fig. 4)

In this scheme, the inverted list for a keyword should include all UID's of the elements which contain it. Thus, the inverted list for the example is as follows:

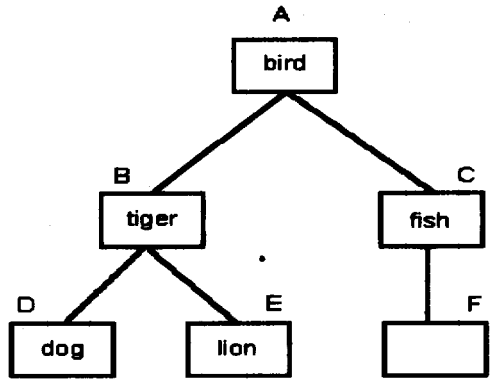


(Fig. 4) Inverted Index with Replication

inverted-list(bird) = {A, B, C, D, E, F}
inverted-list(dog) = {A, B, D}
inverted-list(tiger) = {A, B, D, E}
inverted-list(lion) = {A, B, E}
inverted-list(fish) = {A, C, F}

5.2 Inverted Index with Inheritance(II)

The previous approach has many duplications in the inverted list. However, we need not keep the element identifiers of the internal nodes in the inverted list since we can calculate the ancestor identifiers from the leaf identifiers using the parent function. Moreover, by using the fact that the child nodes of a node can have some keywords in common, we can remove all the duplications of index terms from the inverted list. This is illustrated in (Fig. 5)



(Fig. 5) Inverted Index with Inheritance

We can construct the inverted list for the example as follows:

inverted-list(bird)={A}, inverted-list(dog)={D}
inverted-list(fish)={C}, inverted-list(lion)={E}
inverted-list(tiger)={B}

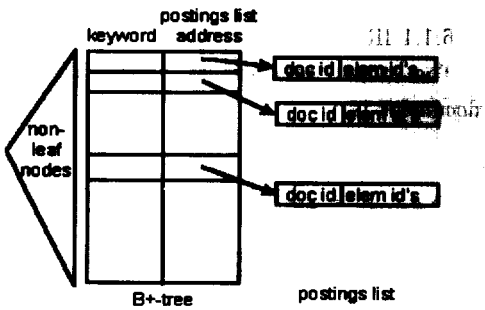
Using the II inverted list, we can access any element using the parent and child function. The index of a node can be calculated as follows:

INDEX = INDEX(itself) ∪ INDEX(ancestors) ∪ INDEX(descendents).

This scheme is much better than the IR scheme since it removes duplications of video element identifiers in the inverted list. Moreover, this scheme supports inheritance of indices between ancestors and descendents. If users create annotations for internal nodes, they are inherited to descendent nodes. Also the keywords belonging to all the child nodes promote to their parent node.

6. Cost Comparison

We analyze the storage requirements and disk access times of the IR and II. The inverted index model which will be used for the analysis is shown in (Fig. 6)



(Fig. 6) Inverted Index Structure

The symbols of 〈Table 2〉 are used in the space and time expressions representing the storage requirements and disk access times respectively.

〈Table 2〉 Symbols and Definitions

symbol	definition
d	degree of B^+-tree
h	height of document tree
k	degree of document tree
l	height of B^+-tree
m	average number of keywords in a node
n_{doc}	total number of documents in database
n_{key}	total number of keywords in database
p	rate of promoted keywords from children to parent
S_{block}	block size in bytes
S_{docid}	document identifier size in bytes
S_{elemid}	element identifier size in bytes
S_{entry}	table entry size in bytes
S_{key}	average keyword size in bytes
S_{ptr}	pointer size in bytes
S_{set}	set size of document variable
t_{random}	random disk block access time
t_{seq}	sequential disk block access time
u	rate of unique index terms in children nodes

6.1 Storage Requirement

The storage requirement of an index structure is the sum of those required for the B^+-tree and postings list.

$$S_{index} = S_{btree} + S_{postings}$$

The B^+-tree has the leaf and internal nodes.

$$S_{btree} = S_{leaf} + S_{internal}$$

Since the total number of keywords in the

database is n_{key}, the number of disk blocks required for the leaf nodes is

$$S_{leaf} = \left\lceil \frac{(s_{key} + s_{ptr}) * n_{key}}{s_{block}} \right\rceil .$$

If we assume that the height of the B^+-tree is l and the degree is d, then the number of disk blocks for the internal nodes is

$$S_{internal} = \sum_{i=0}^{l-2} d^i = \frac{d^{l-1} - 1}{d-1} .$$

The degree d can be calculated as follows:

$$d = \left\lfloor \frac{s_{block}}{s_{key} + s_{ptr}} \right\rfloor .$$

The storage requirement for the postings list for the database is

$$S_{posting} = \left\lceil \frac{n_{key} * S_{id}}{s_{block}} \right\rceil ,$$

where

$$S_{id} = N_{docperkey} * s_{docid} + N_{postperkey} * s_{elemid}.$$

The number of postings per keyword is

$$N_{postperkey} = \left\lceil \frac{N_{postperdoc} * n_{doc}}{n_{key}} \right\rceil .$$

Now we calculate $N_{docperkey}$, the average number of documents per keyword. Here u is the rate of unique index terms among the child nodes which will be indexed in the parent node. The range of u is $\frac{1}{k} \le u \le 1$.

If u is close to $\frac{1}{k}$, the document is very dense, that is, the elements of the document are closely related to each other and have very similar index terms. When u is large, the document is sparse, that is, the elements of the document are not closely related. Assume that the average number of index terms of a leaf node is m. Then the number of index terms of a node at level (h - 1) is $k*m*u$.

The number of index terms of a node at level i is

$$k^{h-i} * m * u^{h-i}.$$

Since the number of nodes at level i is k^{i-1}, the total number of index terms at level i is

$$k^{i-1} * k^{h-i} * m * u^{h-i} = k^{h-1} * m * u^{h-i}.$$

Because the number of index terms of a root node is

$$k^{h-1} * m * u^{h-1},$$

the number of documents per keyword is

$$N_{docperkey} = \left\lceil \frac{k^{h-1} * m * u^{h-1} * n_{doc}}{n_{key}} \right\rceil .$$

Now we calculate $N_{postperdoc}$, the number of index terms per document, for each inverted indexing scheme.

6.1.1 IR

The total number of index terms per document is

$$\begin{aligned} N_{postperdoc} &= \sum_{i=1}^{h} k^{h-1} * m * u^{h-i} \\ &= k^{h-1} * m * \sum_{j=0}^{h-1} u^j \\ &= k^{h-1} * m + k^{h-1} * m * \sum_{j=1}^{h-1} u^j \end{aligned}$$

$$= \begin{cases} k^{h-1} * m + k^{h-1} * m * \dfrac{u*(1-u^{h-1})}{1-u}, & u \neq 1. \\ k^{h-1} * m * h, & u = 1. \end{cases}$$

6.1.2 II

For the analysis of the II, we introduce a parameter p which represents the probability that an index term is promoted from the child nodes to their parent node. The range of p is $0 \leq p \leq 1$.

Then, the number of index terms of a leaf is $m - m*p$.

The number of index terms of a node at level i is

$m*p^{h-i} - m*p^{h-i+1}$.

The total number of index terms at level i is

$k^{i-1} * m*p^{h-i} - k^{i-1} * m*p^{h-i+1}$

because the number of nodes at level i is k^{i-1}.

And the number of index terms of the root node is $m*p^{h-1}$.

Thus, the total number of index terms of a document is

$$N_{postperdoc} = \sum_{i=2}^{h} (k^{i-1} * m*p^{h-i} - k^{i-1} * m*p^{h-i+1}) + m*p^{h-1}$$
$$= k^{h-1} * m - m* \sum_{i=1}^{h-1} p^i * (k^{h-i} - k^{h-i-1}).$$

6.2 Disk Access Time

The disk access time required to retrieve the postings list of a keyword is the sum of the B^+-tree access time and postings list access time.

$T_{search} = T_{btree} + T_{posting}$.

The disk access time of the B^+-tree is

$T_{btree} = l * t_{random}$.

The disk access time of the postings list of a keyword is

$T_{posting} = t_{random} + (N_{blockperkey} - 1) * t_{seq}$,

where the number of disk blocks of the postings list per keyword is $N_{blockperkey} = \left\lceil \frac{S_{posting}}{n_{key}} \right\rceil$.

6.3 Analytical Results

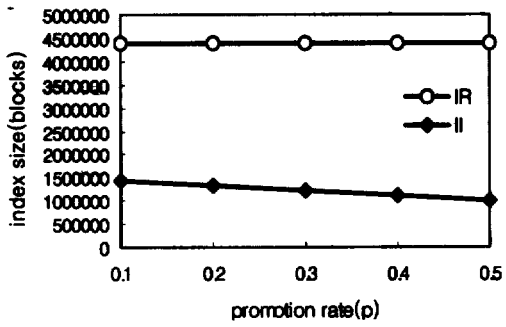
We compare the storage requirements of the inverted indexing schemes by using the parameters as shown in 〈Table 3〉 The disk access times are based on the Conner CP30200 model [6]. We assume that the database has 100,000 video documents with 50,000 keywords.

〈Table 3〉 Parameter Values

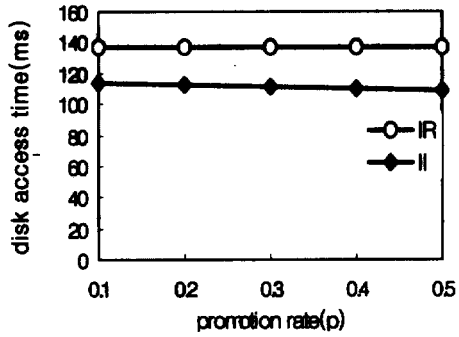
symbol	value	symbol	value
d	64	s_{docid}	4
h	5	s_{elemid}	2
k	5	s_{entry}	4
l	3	s_{key}	12
m	10	s_{ptr}	4
n_{doc}	100,000	s_{set}	50
n_{key}	50,000	t_{random}	19 ms
p	varied	t_{seq}	0.7 ms
s_{block}	1,024	u	0.6

(Fig. 7) illustrates the storage requirements of the IR and II when u is 0.6. It shows that the II has much better performance. It shows that the index size of the II decreases linearly as the value of p increases.

(Fig. 8) illustrates the average disk access times of the two schemes, when u is 0.6 and any arbitrary keyword index is accessed. It



(Fig. 7) Inverted Index Space Requirement

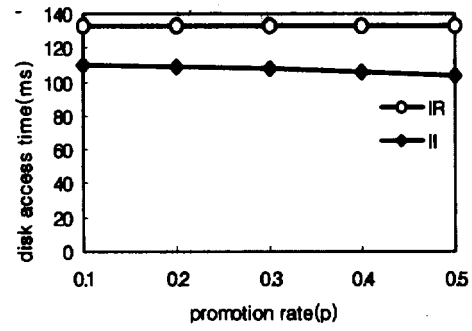


(Fig. 8) Average Disk Access Time

shows that the II has better performance. We also have obtained similar results with other values of u.

6.4 Experimental Results

Experiments have been performed on a SUN SPARC station with a local Conner CP30200 disk. We have experimented in a single user environment after building the IR and II inverted index structures. We have assumed the database contains 100,000 documents with 50,000 keywords, the same parameters as the analysis. (Fig. 9) shows the results of the experiment obtained by accessing the index for a given keyword 100 times each and averaging the access times, when u is 0.6. The results are quite similar to the analytical results.



(Fig. 9) Experimental Average Disk Access Time

7. Conclusions

Our video model represents a video as a tree structure of video elements. Based on the model, a new structure query language which uses path expressions has been defined. Users can freely traverse the logical paths of video documents and retrieve any interested portions of a video or a group of videos with the powerful expression.

We have used unique element identifiers for the video structure representation. This scheme is especially efficient for the evaluation of the branch and list expressions because it does not require database access, compared to the previous schemes which store the structure information into the database as tuples.

We have proposed an efficient inverted index structure supporting structure query for video documents. For performance evaluation, we have presented the performance equations representing storage requirement and disk access time, and have shown the analytical results. The proposed index structure has the inheritance property of the object-oriented concept. By having this property, it can reduce the index space and disk access time considerably.

References

[1] G. E. Blake, et al., "Text/Relational Database Management Systems: Harmonizing SQL and SGML," Proc. of the International Con-ference on Applications of Databases, pp. 267-280, 1994.

[2] P. D. Bruza, "A Novel Aid for Searching in Hypermedia," Proc. of the First European Conference on Hypertext, pp. 109-122, 1990.

[3] D. M. Choy, et al., "Document Management and Handling," IEEE '87 Office Automation Symposium, pp. 241-246, 1987.

[4] V. Christophides and A. Rizk, "Querying Structured Documents with Hypertext Links using OODBMS," Proc. of the European Conference on Hypermedia Technology, pp. 186-197, 1994.

[5] C. Clifton and H. Garcia-Molina, "Indexing a Hypertext Database," Proc. of the 16th VLDB Conference, pp. 36-49, 1990.

[6] Conner Peripherials, Inc., "CP-30200 Specification Summary," 1995.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, The MIT Press, 1990.

[8] D. Harman et al., "Inverted Files," in Information retrieval: Data Structures and Algorithms, W. B. Frankes and R. Baeza-Yates ed., pp. 28-43, Prentice Hall, 1992.

[9] K. Lee, Y. K. Lee, and P. B. Berra, Mana-gement of Multi-structured Hypermedia Documents: Its Data Model, Query Langua-ge, and Indexing Scheme," Journal of Mul-timedia Tools and Applications, vol. 4, no. 2, pp. 199-224, Kluwer Academic Publishers, March 1997.

[10] Y. K. Lee, et al., "Querying Structured Hyperdocuments," Proc. of the 29th Hawaii International Conference on System Sciences, vol. 2, pp. 155-164, Maui, Hawaii, USA, Jan. 1996.

[11] Y. K. Lee, et al., "Index Structures for Structured Documents," Proc. of the First ACM International Confefence on Digital Libraries, pp. 91-99, Bethesda, Maryland, USA, Mar. 1996.

[12] D.-Y. Oh, S. S. Kumar, and P. V. Rangan, "Content-Based Inter-media Synchronization," SPIE Proc. of the Conference on Multimedia Computing and Networking, vol. 2417, pp. 202-214, 1995.

[13] E. Oomoto and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," IEEE Transactions on Knowledge and Data Engineering, 5(4), pp. 629-643, Aug. 1993.

[14] R. Sacks-Davis, T. Arnold-Moore, and J. Z obel, "Database Systems for Structured Doc-uments," Proc. of the International Sympo-sium on Advanced Database Technology and Their Integration, Nara, Japan, Oct. 1994.

[15] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, 1983.

[16] G. Salton, Automatic Text Processing, Addison-Wesley, 1989.

[17] B. Simonnot, "A Cooperation Model for Video Document Retrieval," SPIE Proc. of the Conference on Multimedia Computing and Networking, vol. 2417, pp. 307-317, 1995.

[18] R. Weiss, A. Duba, and D. K. Gifford, "Composition and Search with a Video Algebra," IEEE Multimedia, pp. 12-25, Spring, 1995.

이 용 규

1986년 동국대학교 전자계산학과 (학사)
1988년 한국과학기술원 전산학과 (석사)
1996년 Syracuse University (박사)
1978년~83년 정보통신부 국가공무원
1988년~93년 국방정보체계연구소 선임연구원
1994년~96년 뉴욕 주립 CASE센터 연구조교
1996년~97년 한국통신 선임연구원
1997년~현재 동국대학교 컴퓨터정보통신공학부 전임강사
관심분야 : 멀티미디어 정보검색, 데이터베이스, 하이퍼미디어 시스템, 전자도서관, 소프트웨어 공학