Multimedia Systems (1996) 4: 172–186

Multimedia Systems

The Advanced Video Information System: data structures and query processing

Sibel Adalı, K. Selçuk Candan, Su-Shing Chen*, Kutluhan Erol, V.S. Subrahmanian

Institute for Advanced Computer Studies, Institute for Systems Research, Department of Computer Science, University of Maryland, College Park, MD 20742, USA
e-mail: {sibel, candan, kutluhan, vs}@cs.umd.edu

Abstract. We describe how video data can be organized and structured so as to facilitate efficient querying. We develop a formal model for video data and show how spatial data structures, suitably modified, provide an elegant way of storing such data. We develop algorithms to process various kinds of video queries and show that, in most cases, the complexity of these algorithms is linear. A prototype system, called the Advanced Video Information System (AVIS), based on these concepts, has been designed at the University of Maryland.

Key words: Query processing, video data – Video databases, indexing – Data structures, spatial, modification of – Video data, updating – Advanced Video Information System (AVIS) – Content based search

1 Introduction

During the last few years, the advent of the CD-ROM and the introduction of high-bandwidth communication networks has caused a spectacular explosion in the availability of large video libraries. While a great deal of effort has been invested in problems of how to use bandwidth effectively to communicate large bodies of data across a network, relatively little effort has gone into how to organize, and access, video databases. The primary aim of this paper is to study methods of indexing video databases so as to store and retrieve video data efficiently in of diverse ways. Some example queries to be used for such retrievals are:

Query 1. Find all video frames in which John Wayne appears.
Query 2. Find all video frames in which someone is being murdered.
Query 3. Find all video frames in which Dean Martin appears and Fred Astaire is dancing with Ginger Rogers.
Query 4. Find all people who appear in frames in which Gene Kelly and Ginger Rogers are getting married.
Query 5. Find all video frames showing Ginger Rogers dancing with someone other than Fred Astaire while he is in the same room. (Fred Astaire may not be visible in all these frames, but the fact that he is in the room during that time may be known in advance.)
Query 6. Find all video frames in which Alfred Hitchcock appears after someone is killed.

These are only a few of the examples of the kinds of complex queries that may be asked of a multimedia database system. When designing indexing schemes to store multimedia data, we need to take into account the kinds of queries that will be asked of such a system.

The primary contributions of this paper are the following:

1. We show that certain kinds of existing spatial data structures are adequate for efficiently storing certain kinds of information. In particular, we show that the problem of storing objects occurring in certain frames may be viewed as a problem equivalent to that of storing line segments.
2. We show how we may make use of this "spatial database" intuition to index multimedia data efficiently so that we can handle all the kinds of queries just listed – in particular, we show how a combination of spatial database technology and relational database technology can be "merged" to solve all these kinds of queries efficiently.
3. We describe how updates to information about video data can be implemented efficiently with the data structures just defined.
4. We describe a prototype implementation, the Advanced Video Information System (AVIS), of the data structures and algorithms. The implementation suggests that the theoretical algorithms work very well in practice.

2 Video databases and spatial databases

2.1 A motivating example – a movie by A. Hitchcock

Before we go anyn further, we given an example to be used throughout the paper. In this example, we introduce an

* *Present address:* Information Technology & Organizations Program, National Science Foundation, 4201 Wilson Blvd, Arlington, VA 22230, USA
e-mail: schen@nsf.gov
Correspondence to: S. Adalı

Alfred Hitchcock movie, namely "The Rope" (1948). This movie lasts 80 min.

In this movie, two friends, Philip and Brandon (played by Farley Granger and John Dall) decide to commit the perfect crime. They want to prove that they belong to the privileged group of people who are allowed to kill just for the sake of killing without being punished. They kill their friend David and hide him inside the chest in the living room. To sign their masterpiece, they give a party to which they invite David's girlfriend Janet (played by Joan Chandler), Janet's old boyfriend Kenneth (played by Douglas Dick), and David's father Mr. Kentley (played by Sir Cedric Hardwicke), and David's aunt Mrs. Atwater (played by Constance Collier). These individuals would talk about David, little suspecting his body is in the same room with them. They also invite, as a challenge, their old mentor Rupert Cadell (played by James Stewart) who is known to be very intelligent and suspicious. Rupert will prove worthy of his reputation and will immediately understand the extraordinary circumstances. As the movie progresses, Rupert will keep asking qustions and gather clues to find out what is wrong.

We come back to this example in the following sections and use it as our main example. We have distrubuted clues about the progression and the end of the movie throughout the paper. As the paper progresses, the "careful" reader learns what Rupert learns and how, and what happens at the end.

2.2 Entities in a video database

If we look at the queries given in the introduction, we notice that two types of information are being queried. The first type is a set of *entities* – things of interest to us in the movie. The second type of information is the video frames in which these entities are present. In this section, we list the types of entities in a video that are of interest, and later we explain how the notion of the occurrence of an entity can be mapped onto the spatial domain.

Entities in a video database may have very different structures. For example, if they correspond to the visible object and characters in a video, we can answer the first question given in the introductin. Query 5 involves objects that are present in the scene, but not visible. Queries 3 and 4 deal with entities that have a certain set of attributes (the activity of getting married has the related attributes such as a Groom, Bride, Best Man, etc.), some of which may be specified. Similarly, there may be entities for different media such as audio. We list three types of entities (we do not claim in any way that this is an exhaustive list; the reader is invited to add his or her own entities).

– *video Objects.* These entities are present in the video frames. The video objects may be characters in the movie such as Philip, Rupert, etc., or they may be objects such as the chest in which the body was hidden or the murder weapon. Similarly, objects may be invisible in some video frames, but nonetheless present. For example, the David's body is in the same room as the guests throughout the movie although it is never seen. In fact, this constitutes the ultimate irony in the plot. We restrict ourselves to visible objects for the time being, but we explain how invisible objects can be handled later.

– *Activity types.* Informally speaking, an activity describes the subject of a given video-frame sequence. Thus, for instance, in the movie "The Rope," `murder` is an activity; `giving a party` is also an activity. There is no reason why multiple activities may not occur simultaneously in a video-frame sequence – for instance, during the party someone may `open the chest` containing the body.

– *Event.* An `event` may be thought of as an `instantiation` of an activity type. For instance, the activity type `opening the chest` may refer to two separate events – Philip and Brandon opening the chest to put the body in, and Mrs. Wilson (the housekeeper) opening the chest to put some books in it, are two different events both involving the same activity, `opening the chest`. Note that activity types are general groups containing many events, and they are stored implicitly in the form of a set of events of the same activity type.

  – *Roles.* The events that are of the same activity type are differentiated from each other by the set of objects involved in them. A `role` is a description of certain aspects of an activity. Roles may involve objects – for example, `victim` and `murderer` are roles in the activity `murder`. Roles may also be descriptions, such as the `murder motive` and `murder weapon`.

  – *Teams.* A `team` is a set of objects/descriptions that jointly describe an event, i.e., they are the instantiation of the roles in an activity type. Thus, for instance, if we talk of the event `murder` of David by Philip and Brandon, then the team involved consists of David in the role `victim`, and Philip and Brandon, both in the role `murderer`. The rope plays the role of the `murder weapon`, while `proving that they committed the perfect crime` is the role `murder motive`. Each of these members of a team is called a `player`. Note that conceptually video objects may be considred a special activity type in which the teams are always empty. Because of this, and as they are the entities accessed most frequently, we have decided to represent them separately.

Example 1. For our example, we have chosen the following video objects and activity types with their corresponding events (the numbers in parentheses show the unique number given to them in our video database.)

Video objects

1. Philip
2. Brandon
3. Mrs. Wilson
4. Rupert
5. Janet
6. Kenneth
7. Mr. Kentley
8. Mrs. Atwater

9. The chest
10. The rope (murder weapon)

Events.

1. Murdering of David with a rope by Philip and Brandon.
2. Giving a party where Philip and Brandon are the hosts and other people (objects 4 through 8) are the guests.
3. There are three events of the activity type `opening the chest`. The actors are Philip and Brandon in the first of these events when the body is placed in the chest.
4. Mrs. Wilson opens it to put some books, but she does not see the body.
5. Rupert opens it at the end of the movie to see if the body was really there as he suspected.

Events 6–13 are of type "find clues", and they are given later in the paper.

2.3 Frame sequences

In the area of video databases, we observe that video data can be classified as follows. A body of *raw video data* (RVD) is a set of entities called *frames*. For the purposes of this paper, a frame may be thought of as a piece of video of short duration. In addition to the RVD a set of "entities" (ENT) consists of objects that are of interest within the domain of a given RVD. An association map, λ, specifies which objects occur in which video frames. Marcus and Subrahmanian (1994) have, in a series of papers, specified how multimedia database systems can be formally modeled along the lines we have briefly outlined. To specify the association map formally, we need some elementary definitions and assumptions.

Assumption. We assume that the set of frames of RVD, is the set $\{1, \ldots, n\}$ for some arbitrarily chosen, but fixed, integer n. Likewise, we assume that the set of objects is also the set $\{1, \ldots, k\}$ for some arbitrarily chosen, but fixed, integer k. There is no loss of generality in making this assumption; it is convenient in providing a formal mathematical framework for reasoning about multimedia systems.

A *frame-sequence* is a pair $[i, j)$ where $1 \leq i, \leq j \leq n$. $[i, j)$ Represents the set of all frames between i (inclusive) and j (noninclusive). In other words, $[i, j) = \{k \,|\, i \leq k < j\}$; i is said to be the *start* and j, the *end* of the frame sequence $[i, j)$.

We can define a partial ordering, $\sqsubseteq$, on the set of all frame sequences as follows: $[i_1, j_1) \sqsubseteq [i_2, j_2)$ if $i_1 < j_1 \leq i_2 < j_2$. Intuitively, $[i_1, j_1) \sqsubseteq [i_2, j_2)$ means that the sequence of frames denoted by $[i_1, j_1)$ precedes the sequence of frames denoted by $[i_2, j_2)$. As usual, we use $[i_1, j_1) \sqsubset [i_2, j_2)$ to denote that $[i_1, j_1) \sqsubseteq [i_2, j_2)$ and $j_1 \neq i_2$.

A set X of frame sequences is said to be *well ordered* if:

1. X is finite, i.e., $X = \{[i_1, j_1), \ldots, [i_r, j_r)\}$ for some integer r, and
2. $[i_1, j_1) \sqsubseteq [i_2, j_2) \sqsubseteq \ldots \sqsubseteq [i_r, j_r)$.

For example, the set $X = \{[1, 4), [9, 13), [33, 90)\}$ is a well-ordered set of frame sequences because $[1, 4) \sqsubseteq [9, 13) \sqsubseteq [33, 90)$. However, $X' = \{[1, 7), [5, 8)\}$ is not well-ordered because $[1, 7) \not\sqsubseteq [5, 8)$ and $[5, 8) \not\sqsubseteq [1, 7)$.

A set X of frame sequences is said to be *solid* if:

1. X is well ordered and
2. There is no pair of frame sequences in X of the form $[i_1, i_2)$ and $[i_2, i_3)$.

For instance, if we take $X = \{[1, 5), [5, 7), [9, 11)\}$, then X is not solid, even though it is well ordered. It may be converted into an equivalent solid set of frame sequences by replacing X with $\{[1, 7), [9, 11)\}$, which is solid. Intuitively, the requirement of solidity says that if we have two abutting frame sequences $[i, j)$ and $[j, k)$, then these should be merged to form the frame sequence $[i, k)$.

An *association map* is a function λ such that for each entity $ent \in$ ENT, $\lambda(ent)$ is a solid set of frame sequences. Intuitively, $\lambda(ent) = \{[i_1, j_1), [i_2, j_2)\}$ means that entity ent occurs in all frames in the frame sequences $[i_1, j_1)$ and $[i_2, j_2)$.

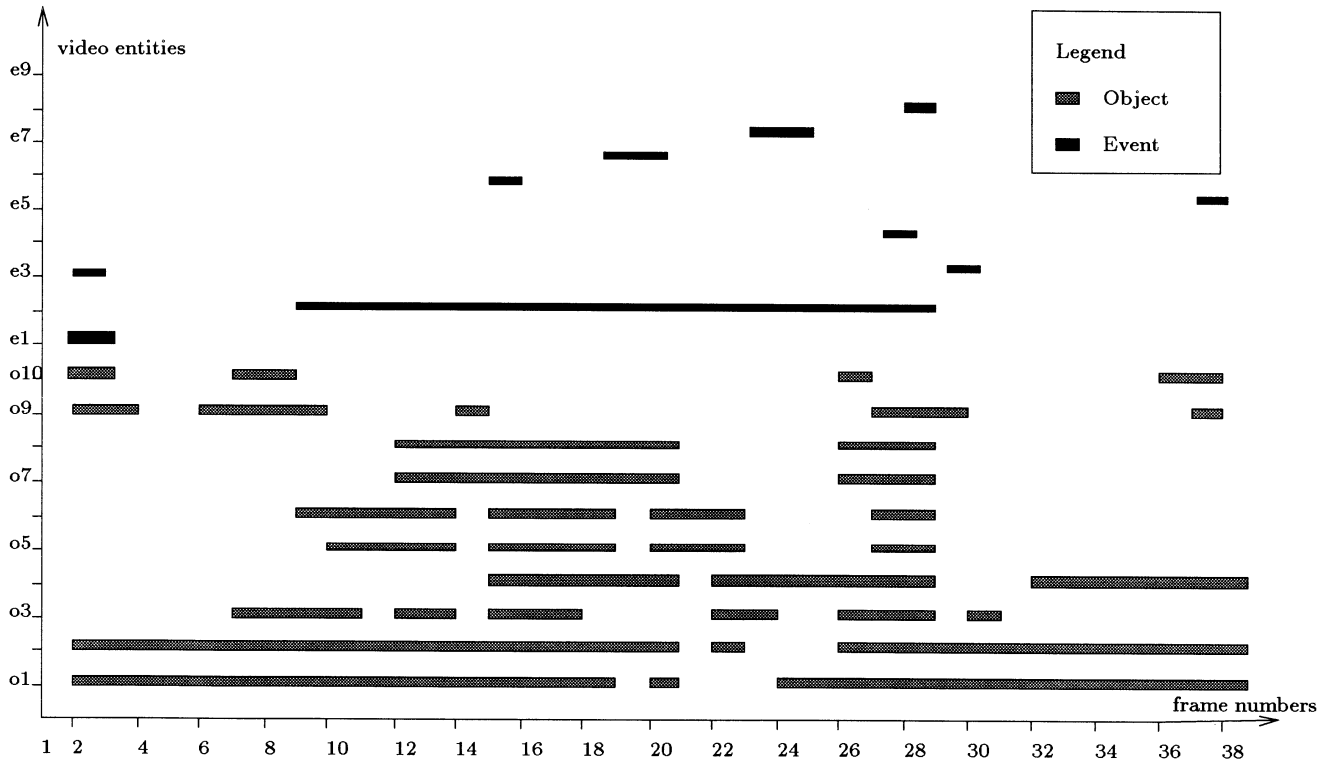
Example 2. Recall the movie "The Rope". For purposes of annotation, we have divided the movie into 40 frames of 2 min each. Note that the granularity of this division is much coarser than the ideal case. We have determined where each object and each event occurs, thus specifying the association map for this movie. Object 1 (Philip) appears in most of the movie, hence the association map for Philip is $\lambda(o1) = \{[2, 19), [20, 21), [24, 40)\}$, which indicates that Philip is seen in frames $2, \ldots, 18, 20$ and $24, \ldots, 39$. Object 7 (Mr. Kentley) and object 8 (Mrs. Atwood) come to the party together, they stay together, and they leave together. Hence, $\lambda(o7) = \lambda(o8)$.

Other examples of the frame sequences and the objects/events that appear in them are: Brandon: $\lambda(o2) = \{[2, 21), [22, 23), [26, 40)\}$; Mrs. Wilson, the housekeeper: $\lambda(o3) = \{[7, 11)$,$[12, 14)$, $[15, 18)$, $[22, 24)$,$[26, 29)$,$[30, 31)\}$; Rupert: $\lambda(o4) = \{[15, 21), [22, 29), [32, 40)\}$; Philip and Brandon opening the chest: $\lambda(e3) = \{[2, 3), [29, 30)\}$; Mrs. Wilson opening the chest: $\lambda(e4) = \{[27, 28)\}$; Rupert opening the chest: $\lambda(e5) = \{[37, 38)\}$, etc.

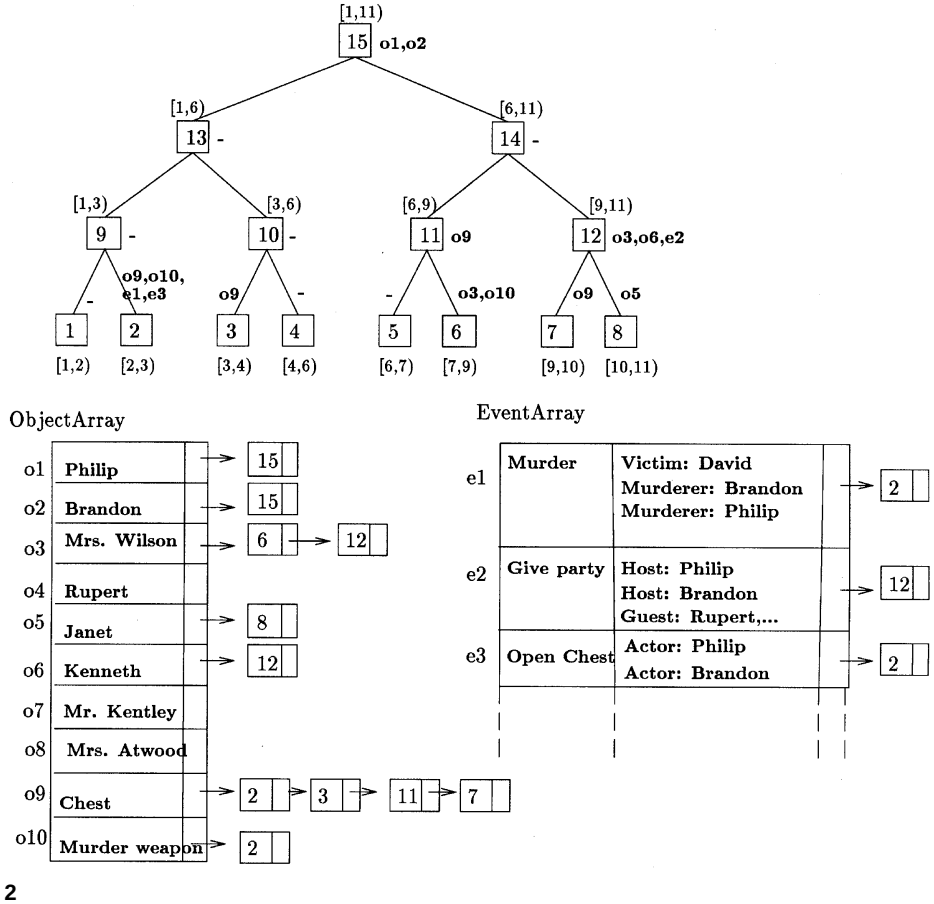
2.4 Association maps and segment trees

In this section, we show that association maps correspond exactly to line segments on the x-axis of the Cartesian plane, and hence, association maps can be stored by any method for storing such collinear line segments, such as the well-known segment tree (Samet 1989)). As an example, consider the association map depicted in Fig. 1 for the movie "The Rope" according to the definitions given earlier in this section. In the figure, we see two types of entities: video objects and events. As can be seen from the figure, we can represent the set of all line segments associated with RVDas a segment tree. The one difference between these segment trees and ordinary segment trees (Samet 1989) is that a single object name can label multiple line segments. To differentiate between the two, we refer to our data structure as a frame segment tree.

The observation that association maps correspond to line segments is not neww – it has been made by several individuals in the multimedia community. In this diagram, the fact that video object 1 (Philip) appears in video frames



1



2

Fig. 1. The association map for "The Rope"

Fig. 2. The object-array-linked segment tree

$2, \ldots, 18$, 20 and $24, \ldots, 39$ is represented by the three shaded horizontal lines associated with object 1. One of the key observations in our paper is that this object-frame map is just a set of line segments, all of which are horizontal. Consequently, the frames in which a given object appears may be viewed as a set of line segments on the x-axis, obtained by projecting the shaded line segments shown in the figure onto the x-axis. Hence, an appropriate data structure for this problem would contain:

- A segment tree representing the set of all line segments in RVD
- Additional arrays, such as OBJECTARRAY, EVENTARRAY and ACTIVITYARRAY, which provide additional indices to the nodes in the tree denoting the locations of the corresponding data

Figure 2 specifies this data structure for the interval $[1, 11)$, as applied to the association map shown in Fig. 1. We now explain the construction of the data structure shown in Fig. 2.

Frame-segment-tree representation of video frames

1. Each node in the frame segment tree represents a *frame sequence* $[x, y)$ starting at frame x and including all frames up to, but not including, frame y.
2. The number inside each node can be viewed as a pointer to that node.
3. The set of numbers (in boldface) placed next to a node are the identification (id) numbers of video objects and events that appear in the entire frame sequence associated with that node. Thus, for example, if a node N represents the frame sequence $[i, j)$ and object o occurs in all frames in $[i, j)$, then object o labels node N (unless object o labels an ancestor of node N in the tree).

The OBJECTARRAY. The object array's ith element denotes video object i (denoted by oi). Associated with any element of this array is an *ordered-linked list* of pointers to nodes in the frame segment tree. For example, Fig. 2 shows that the linked list associated with object 3 (o3) contains two pointers – to nodes 6 and 12. These are the nodes in the frame segment tree that are labelled with object number 3. Nodes 6 and 12 represent, respectively, the frame sequences $[7, 9)$ and $[9, 11)$. The nodes are stored in the order shown in Fig. 2 because

$[7, 9) \sqsubseteq [9, 11)$.

The astute reader may now wonder: the frame segment tree, by itself, contains all the information necessary to capture the association map. Therefore, why is it necessary to store an object array? The answer is that queries may be formed on multiple keys, and the object array facilitates efficient handling of certain kinds of queries. Consider elementary membership queries. Here, we wish to find all frames in which object o occurs. Using the object array makes it a relatively simple matter – we just return the entire list associated with video object o; locating the entry for o is a constant time operation in this case.

The EVENTARRAY. As in the case of the OBJECTARRAY, the event array stores the frame sequences, in the form of links to the frame segment tree, for each different event. Events are uniformly numbered from e1 to eN. For each event, the following information is stored: the activity type, the team as a list of pairs of the form (role: player) and finally, a list of tree nodes that indicates the frame sequences in which this event occurs.

The ACTIVITYARRAY. This is another index that simply stores, for each activity type, the list of events of that type. This facilitates queries about a certain activity type as opposed to a certain event of that type. Similarly, the ROLEARRAY simply lists the name of the roles, since the roles are also stored by their unique id number in the video database. In addition to these, we assume that we have hash tables for objects, activity types, and roles that map their names to their unique id numbers. We do not give details of these tables since they are only used to locate the id numbers.

3 A full description of video databases

We are now in a position to describe a video-database formally. First, we assume that OBJ is a set of video objects and EVT is a set of events found in the video database. Hence, ENT = OBJ $\cup$ EVT (ENT is the set of all entities). The association map λ maps elements of ENT onto sets of frame sequenes. For each event $e \in$ EVT, a many-to-one mapping $\aleph$ maps each event to an activity type. For example, in our movie example, $\aleph$ (e1) = murder.

Next, we assume that, given an activity type A, a set ROLE(A) is associated with it. The members of the set ROLE(A) are strings denoting the names of the roles associated with activity A. Thus, for example, ROLE(murder) = {victim,murderer,murdermotive}, and ROLE ({giving a party} = {host,guest}.

Given any event E of activity type A, we assume that there is a mapping PLAYERS that maps ROLE(A) to the set OBJ $\cup$ EVT $\cup$ STR; STR is the set of all possible strings. The range of the map PLAYERS specifies the team involved in the event.

Definition 1. A *video database* is a 9-tuple

(RVD, OBJ, EVT, λ, ACT, $\mathfrak{R}$, $\wp$, ROLE, PLAYERS)

where

- RVD is a set of integers $\{1, \ldots, n\}$ for sone n.
- OBJ is a set of *objects*.
- EVT is a set of *events*.
- λ is an association map that assigns a solid set of frame sequences to each entity $ent \in$ (OBJ $\cup$ EVT).
- ACT is a set of *activity types*.
- $\mathfrak{R}$ is a set of *roles*.
- $\wp$ is a map that assigns an activity type to each event.
- ROLE is a map that takes a frame sequence $[i, j) \in$ EVT $\cup \bigcup_{o \in \text{OBJ}} \lambda(o)$, an activity A in $\wp([i, j))$, and a role in $\mathfrak{R}$(A) as input and assigns a member of OBJ as output.
- PLAYERS is a map that takes an event and its activity type as input and returns a mapping from the roles of the activity to the entities in the database and to strings as output.

4 Data structures

In this section, we define data structures that are necessary in order to store the video data that we are interested in dealing with. There are four types of structures that need to be defined: a *frame segment tree* that allows us to specify which events (and their relevant players, teams, etc.) and video objects occur in which segments of video; an OBJECTARRAY that allows accessing video-segments using objects as keys, an ACTIVITYARRAY that allows accessing video-segments using activity types as keys, and an EVENTARRAY that facilitates accesses using events as keys. Of these, the key component is a special kind of segment tree.

4.1 Frame segment trees

In this section, we define a special kind of segment tree called a frame segment tree that is used for multimedia data. The structure of nodes in the frame segment tree can be defined as follows. Note that in this data structure, the players are restricted to video objects. It is straightforward to augment the data structures to allow players of any type.

```
type treenode = record of
   start    : integer;  /* starting frame number */
   finish   : integer;  /* ending frame number */
   objlist  : ^objnode ; /* list of objects
              associated with the node */
   evtlist  : ^evtnode;  /* list of activities
              associated with the node */
   lchild   : ^treenode ; /* left child */
   rchild   : ^treenode;  /* right child */
   end;

type objnode = record of
   objid  : integer ; /* object number */
   next    : ^objnode; /* next node in list */
   end;

type evtnode = record of
   evtid : integer; /* activity number */
   next   : ^evtnode; /* next node
   in activity list */
   end;
```

Assumption. We assume that, for any node N of type `treenode`, N.`objlist` and N.`evtlist` are maintained in ascending order with respect to the object number and the event number, respectively. Thus, if we consider node 9 in Fig. 2, the `objlist` associated with this node is stored in the order **o3,o10**.

4.2 The other arrays

The OBJECTARRAY, the ACTIVITYARRAY, and the EVENTARRAY can be defined as arrays with elements having the record structures `objectrec`, `activityrec`, and `eventrec`, respectively. We assume that `Val1,Val2, Val3,Val4` are constants.

```
type objectrec = record
     name   : string; /* name of object/
     frames : ^frameseqlist;
              /* pointer to list of tree nodes */
     end;

type activityrec = record
     name   : string;
     events : ^evtnode;
     end;

type eventrec = record
     acttype:  integer;
     teamlist: ^playernode;
     frames : ^frameseqlist;
end;

type playernode = record of
     player: integer; /* object number */
     role  : integer; /* role number */
     next : ^playernode; /* next player
     in the team */
     end;

type frameseqlist = record
     segments : ^treenode; /* points to a
     tree node */
     next : ^frameseqlist; /* next node in list */
     end;

eventarray : array[1..Val1] of eventrec;
activityarray : array[1..Val2] of activityrec;
objectarray : array[1..Val3] of objectrec;
rolearray: array[1..Val4] of string;
```

Using this formal description of the data structures, we continue with:

1. Listing of the elementary queries that can be posed to our system
2. Development of the algorithms to execute these queries
3. Complexity analysis of the algorithms for each query
4. A formal rendering of algorithms to update the data structure efficiently when new entities in existing videos become important.

5 Query processing

In the preceding section, we specified a data structure for storing video information. In this section, we try to show that this data structure facilitates the efficient execution of various types of queries. We give examples of such queries and explain how each can be answered with our data structure. In addition to developing algorithms for handling such queries, we analyze the algorithmic complexity of these procedures.

5.1 Elementary object queries

This is a query of the form: “Given the name of an object, find all video frames in which a given object occurs.”

Example. “Find all the video frames in which the murder weapon (object 10) is seen.” This query returns the answer video frames 2, 7, 8, 26, 37, 38, or it may simply return a set of frame sequences capturing these video frames, namely, { [2,3),[7,9),[26,27),[37,39)}. Similarly, the query “find all

video frames in which Brandon is seen" returns the answer { [2,19),[20,21),[24,40)}.

Method. The query can be processed by first finding the id of the object in question in the OBJECTARRAY with the name given in the query and the hash table mentioned earlier. Then, follow the link OBJECTARRAY.frames and create a set of frame sequences corresponding to the start and end points of the tree nodes for this object. Finally, merge the frame sequences to obtain a solid set.

Complexity. As the object in OBJECTARRAY can be found in a constant time, the time required is proportional to (i.e., linear in) the number of segments in which the object appears, which is also the size of the output.

5.2 Elementary activity-type queries

This is a query of the form: "find all the video frames in which events of a given activity type occurs."

Example. "Find all the video frames in which someone is opening the chest." There are three events corresponding to this activity type, in which the actor is either Philip and Brandon, or Mrs. Wilson, or Rupert. (In practice, activity types are more general, such as `open an object`, but we use "open the chest" for our example movie.) Hence, we must find the frame sequences for all these events and merge them. Finally, we get the answer { [2,3),[27,28),[29,30),[38,39) }.

Method. The query can be solved by first locating all the events corresponding to the given activity type. This can be done by examining ACTIVITYARRAY and the hash table, then following the list pointed to by ACTIVITYARRAY.events. The set of frame sequences for all such events are obtained one by one by following the links in the EVENTARRAY and merging the results into solid sets of frame sequences. These sets of frame sequences are then merged into a final solid set to give the final output.

Complexity. As in the case of elementary object queries, the time required is proportional to (i.e., linear in) the number of segments associated with the events of the activity type under consideration.

5.3 Detailed activity queries – event queries

This is a query of the form: "find all video frames in which one of a given set of events occurs. The events are specified by the activity type and the roles of specific objects involved in this activity". Since the players provided in the query can be a partial list of all the players of an event, more than one event may satisfy such a query.

Example. "Find all the video frames in which Rupert is given a clue by Philip." Although both Philip and Brandon are both cold-blooded murderers, Brandon is the brains behind the team with Philip only being his helper. Philip is a very nervous person, and he inadvertently gives away some clues to Rupert, who closely scrutinizes his every move. First, when Brandon tells a story about his strangling a chicken when they were kids, he shows an unexpected reaction. Rupert, however, knows this story is true (cf. frame 17). Second, Brandon gives some books to Mr. Kentley and ties them up with the rope – the murder weapon. When Philip sees this rope, he is petrified (frame 26), and of course he is questioned by Rupert about these events. Hence, the original query should return: $\{[17, 18), [26, 27)\}$.

Method. This query is similar to an elementary activity query except that the search to locate relevant events is more complex. For this, first the activity type in question is located, for example, `finding clue`. Then, all the events listed under this activity type are searched. For all such events, we check to see whether the teams contain all the necessary players, for exxample, `finder:Rupert` and `giver:Philip`. Then, for all these events, the link to the frame segment tree is followed, and the solid set of frame sequences is formed. Finally, all these sets are merged to give the final result.

Complexity. The time required is proportional to the number of segments in the events satisfying the conditions and the cost of checking each event of a given activity type for role/object pairs in the query. The cost of this check is the number of role/object pairs in the events of given activity type.

5.4 Object-occurrence queries

This is a query of the form: "find all objects that occur in a given set of frame sequences".

Example. "Find all the objects that are present in frame sequence [10,29)." Incidentally, this frame sequence corresponds to the whole duration of the party in the movie. Hence, it gives us everybody involved in the party one way or another. This query returns all the video objects in the database including the chest and the murder weapon. The reader may notice that if the invisible object `David's body` were stored in the database, it would be returned by this query. Ironically, it is also part of the party.

Method. The query can be solved by searching the frame segment tree starting from the root for the given set of frame sequences. If the node being visited intersects any of the frame sequences in the set, all the objects stored in this node are added to the output set of objects, and then both the left and the right children of the node are visited. It is possible to split the set of frame sequences for the children so that only those sequences that may possibly intersect the corresponding child are included in that call. Each time a set of new objects is added to the output list, duplicates need to be removed. This is a costly operation. Using a bag that clusters objects is a step towards reducing the complexity of this operation.

Complexity. As this information can be computed via the segment tree, the cost involved is proportional to the number of those nodes in the segment tree whose parents' intervals overlap a given frame sequence in the query times log (*number of input frame sequences*) plus the number of objects in each of those nodes. It cannot be computed any faster because each such node must be checked. (The log

factor is the cost of checking whether a given node's interval overlaps with a frame sequence in the query).

5.5 Activity-type occurrence queries

This is a query of the form: "find all the activities that occur in a given set of frame sequences".

Example. "Find all the activities that occur in the first 30 frames." This gives the activities of murder, opening the chest, giving a party, playing the piano.

Method/Complexity. This is very similar to the object occurrence query. The tree is searched by the same algorithm. This time the events are collected, and, at the end of the search, the collected events are grouped into their respective activity types. This list is returned.

5.6 Conjunctive queries

If one examine the queries given, one can see that they can be grouped into a few query types, namely, (1) queries that take a specification as input and return a set of frame sequences as output, (2) queries that take a set of frame sequences as input and return a set of objects, (3) queries that take a set of frame sequences as input, but returns a set of events as output, and (4) some other queries that we have not discussed; they take the same types of input as 2 and 3, but they return activity types, role of a specific object, etc., as ouput.

An example of such a query is the query that takes a set of frame sequences as input and returns all the objects and events in those frames as output. The query that takes an object and an event as input and returns the set of all frames in which this object is seen and the event occurs as output is also possible.

Example. "Find all the frames in which people are eating while the chest can be seen." This actually happens in the movie, since Brandon moves the appetizers from the table to the top of the chest before the guests arrive. This makes Rupert even more suspicious when he hears this in his talk with Mrs. Wilson.

Method. The algorithm depends on the specific query posed. For the conjunction given, we execute the simple object and event queries for the relevant parts of the query. Then we intersect these frame sequences to obtain the result. Note that the frame sequences returned by these queries are ordered, hence the intersection operation is not very costly. This approach is particularly useful since it makes use of the previous code.

Complexity. The time required equals the sum of the time required for each component plus the cost of taking the intersection of the answers. If answers from components happen to be sorted, their intersection can be found in the order of their size. Otherwise taking the intersection is $O(n \log(n))$ where n is the sum of the sizes of the individual queries.

5.7 Compound queries

Compound queries involve the relationship between various objects and events. In this section, we only consider nontemporal queries. Suppose we consider the query "find all the objects present in the video when a certain event (specified by the activity type and a team) occurs". In general, answers to compound queries are found by directing the output of one function as input to another function.

Example. "Find all the people present in the scene when Brandon and Rupert discuss whether murder is a privilege reserved for a small group of people." This happens in frames 19 and 20. Hence, the objects seen in these frames are Philip, Brandon, Rupert, Kenneth, Janet, Mr. Kentley, and Mrs. Atwater.

5.8 Other types of queries

As we discuss earlier in the paper, we may have various types of video objects, such as invisible objects. We may also have audio events, i.e., events referring to the dialogues in the movie, such as someone singing. The queries involving these objects are very similar to the ones given here since, at the end, invisible objects are objects, and audio events are still events with the same properties. We do not specify how these can be stored; however, one can easily think of rather straightforward extensions of our data structure to handle these objects. The query processing algorithms can then be modified slightly to process queries involving these entities, as well as the previous ones. Another query type could be a general query in which the user specifies a complex condition, such as the combination of several of the queries already mentioned. Solving these queries is beyond the scope of this paper.

In addition, one can ask temporal queries, such as query 6 in the introduction. The semantics of such queries has not yet been specified by the authors, and will be addressed in future work. For example, operators such as *after*, *right after*, *before*, and *overlaps* should be included (Iino 1994). In addition to this, the semantics of operators such as *before* and *after* must be established clearly. Consider the query: "find all the events in which Rupert is involved after he finds out that Brandon (murderer of David) told Kenneth (Janet's old boyfriend) that he feels Kenneth may have better chances with Janet (David's girlfriend) after all." If that the specified event happens in two places, then *after* may mean after the first occurrence or after the last occurrence of this event. We have deferred the treatment of such queries to a future paper.

6 Updates

In the real world, it is likely that a video database system organized along the lines described thus far will change over a period of time. We have classified all changes into the following two categories.

1. *Insertion or deletion of an entity.* Examples of such insertions are the insertion of a new object, of a new player into an event. Similarly, some objects, events, etc., with all the

relevant information stored in the video database may also have to be deleted.

There may be various reasons for such operations. For example, it may be the case that an object o occurred in some frame f. However, the significance of o may not have been realized earlier, and therefore, the need to store o and the fact that it occurred in f may not have been realized. It is not far fetched to assume that, given the annotation of the movie "The Rope," an experienced movie watcher would immediately ask, "Find me the frames in which Hitchcock appears personally." Since Hitchcock is not an annotation in the video, this query cannot be answered. Hence, someone may decide later on that this object needs to added to the database.

Another reason for such a change in the movie database is the possibility that some of the entities were wrongly annotated. If an object has been identified incorrectly, it may have to be deleted altogether and re-inserted into the data structure. Consider the case in which an Elvis "look-alike" has been categorized as Elvis Presley, but later inspection shows that he is not, in fact, Elvis Presley, so that this object `Elvis Presley` must be deleted.

Modification of frame sequences. Since the databases may contain incomplete information, we may find that an object or an event actually occurs in additional frames that were not in the initial list. Thus, we need to modify the database by inserting new frame sequences into the data structures. Similarly, we may find that some objects (events) have been mistakenly listed as occurring in certain frames even though they do not, in fact. For this reason, we may need to delete, from the frame segment tree, some of the frame sequences associated with an object, as well as the corresponding array(s).

It is easy to see that some updates are pretty straightforward. In particular, some updates are of a "relational" nature and do not involve temporal properties. The updates that require modifications to the frame segment tree can be broken up into more elementary operations. Such updates include: (1) inserting a set of frame sequences into the frame segment tree and the `OBJECTARRAY` for a specific object, (2) deleting a set of frame sequences for a specific object from the database, and (3–4) inserting and deleting for an event. The updates listed earlier in this section can be encoded by treating these operations as the elementary ones. For example, an object can be deleted by locating it, retrieving the set of frame sequences for it, and then deleting all of these sequences. In the following sections, we give the first two of these operations in detail. Subsequently, we quickly outline the methods for implementing operations 3 and 4, which are similar to operations 1 and 2.

6.1 Insertion of a set of frame sequences for an object

For this operation, the name of the object being inserted and the solid set of frame sequences in which this object appears are given as input. The data structure is modified in such a way that the given object appears in the specified frames.

Method

1. If the given object is not in `ObjectArray`, then create an entry for it in `ObjectArray`.
2. For each frame sequence in the input list do:
   a) Put the frame sequence into the tree and mark the necessary tree nodes with the id of the object. This step is further described later.
   b) For each tree node N marked by the object do:

      Insert a node F into the `frames` list of the object in `ObjectArray` so that F has a pointer to the tree node N. Note that the `frames` lists must be preserved in sorted order.

Recall that we restrict the boundaries of the nodes in the frame segment tree to the start and end points of frame sequences in which entities occur. Hence, a frame sequence that is being inserted may not fit into any of the nodes in the tree completely. In this case, we must split the boundaries of some nodes. Let the frame sequence we are inserting into the tree have the boundaries $[a, b)$. There are three cases to consider:

- *Case 1.* There is a leaf l in the tree with boundaries $[c, d)$, where $a <= c < b$ and $d > b$. In this case, node l will have two new children with the boundaries $[c, b)$ and $[b, d)$.
- *Case 2.* There is a leaf l in the tree with boundaries $[c, d)$, where $c < a < d$ and $d <= b$. In this case, the two new children of the node l will have the boundaries $[c, a)$ and $[a, d)$.
- *Case 3.* There is a leaf l in the tree with boundaries $[c, d)$, where $c < a$ and $b < d$. In this case, l will have two children. Child 1 will have the boundaries $[c, a)$ and the child 2, $[a, d)$. In addition to this, child 2 will have two children, also with the boundaries $[a, b)$, and $[b, d)$.

Note that as the boundaries of the parent node are not changed in any of these cases, none of the other objects pointing to the parent node is affected by this change in the tree. Hence, the updates can easily be accomplished. However, the tree may become unbalanced if there are many insertions of this type. Standard tree balancing techniques can easily be adapted to balance such trees when necessary.

Complexity. This task can be performed in time $O(nh)$ where h is the height of the segment tree and n is the cardinality of the (solid) set of frame sequences associated with the object being inserted.

6.2 Deletion of a set of frame sequences for an object

As in the case of insertion, this operation takes the name of the object, and the solid set of frame sequences (in which this object appears) that are to be deleted as input. The algorithm for this operation is:

1. If the object is not in `ObjectArray`, halt and report failure.
2. Otherwise, for each frame sequence $[a, b)$ in the input list do:

a) If there is a node N in the segment tree with a frame sequence of the form $[c, d)$ in the `frames` list where $c <= a$ and $b <= d$ then:
   i. Remove the object from the `objlist` of the corresponding tree node.
   ii. Remove N from the `frames` list.
   iii. Insert the object into the tree with the frame sequence set {[c,a),[b,d)}. Note that if $c = a$ ($b = d$) then $[c, a)$ ($[b, d)$) is omitted from the set.

b) If there is a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c > a$ and $b <= d$ then:
   i. Remove the object from the `objlist` of the corresponding tree node.
   ii. Remove N from the `frames` list of the corresponding `ObjectArray` node.
   iii. Insert the object into the tree with frame sequence set {[b,d)}. Note that if $b = d$ then $[b, d)$ is omitted from the set.

c) If there is a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c <= a$ and $b > d$ then:
   i. Remove the object from the `objlist` of the corresponding tree node.
   ii. Remove N from the `frames` list of the corresponding `ObjectArray` node.
   iii. Insert the object into the tree with the frame sequence set {[c,a)}. Note that if $c = a$ then $[c, a)$ is omitted from the set.

d) If there is a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c > a$ and $b > d$ then:
   i. Remove the object from the `objlist` of the corresponding tree node.
   ii. Remove N from the `frames` list.

3. If all the frame sequences in `frames` have been removed, release the corresponding `ObjectArray` slot.

6.3 Insertion/Deletion of a set of frame sequences for an event

In this case, an event id is provided as the input. An event may have been identified by its activity type and its team, but for the correct execution of this operation, the unique id of the event in question must be provided. After the event is identified and the set of frame sequences are given, the operations will be exactly the same, except that they will access the `EVENTARRAY` and the `ACTIVITYARRAY` instead of the `OBJECTARRAY` as in the previous cases. We do not give the details of these operations here.

6.4 Other operations

We briefly mention other simple updates that can be performed on the video database.

Player insertions. To insert a new player into the database, the name of the object, the role of the object and the id of the relevant event is given as input. Then the corresponding event e is located in `EventArray`, and a new node p is inserted into the `players` list of e with the id of the object (found in the hash table) and the given role.

Player deletions. To delete a player from certain events in the video database, the name and the role of the player, together with the corresponding event id, are given as input. The entry for the event is then located in `EventArray`. The player playing the given role is then searched for in the `players` list with its id. If it is found, the node containing this information is deleted from the list of players.

6.5 Storing multiple videos

Thus far, we have focused on storing a single video. In the future, users will need the ability to query a video library for certain objects. The user may ask queries such as: “Find all the video frames in your database (not necessarily from a specific movie) in which Alfred Hitchcock appears.” To handle such queries, it is possible to maintain a global list of objects, events, activity types in a kind of array structure like the `OBJECTARRAY` and the other similar arrays. This array will contain a list for each object, for example, a list of videos that contain that object. Similarly, global event arrays, global role arrays, etc., will be maintained as in the case of a single video. Hence, a query such as the one just mentioned can be answered by first locating relevant videos, and then directing an elementary object query to the indices associated with these individual videos. This has the possible advantage of various processes sharing a global hash-name server for all entities in the database. Our system can implement this capability quite easily.

7 Implementation of the Advanced Video Information System (AVIS)

The algorithms described in this paper have been implemented in a prototype system, AVIS. Subsequently, AVIS has been hooked up to a much more powerful system for software integration, the Heterogeneous Reasoning and Mediator System (HERMES) (Subrahmanian 1995). An advantage of hooking up AVIS and HERMES is that all the parsing capabilities of HERMES are automatically available to AVIS. Furthermore, as HERMES allows complex queries that integrate data and reasoning across multiple domains, it is now possible to ask queries of the form: find all frame sequences F and an actor A such that actor A also appeared in some movie M, and such that actor A appears in the frame sequences F. This query may require access to a relational database (say, INGRES) that specifies which actors occur in which movie, as well as access to the video content of the “The Rope”, provided by AVIS. The video content of “The Rope” can be stored with the data structures described in this paper.

We now quickly describe the implementation of AVIS, followed by the integration of HERMES and AVIS.

AVIS. AVIS implements all the query processing algorithms described in this paper. It consists of approximately 4000 lines of C code, and runs on the UNIX/XWindows platform. The basic functionality of AVIS is captured by the following predefined predicates, each of which access functions (having the same name as the predicate described here).

1. `event_to_frames(Event, Framelist)`. Given an event (which is a complex type), this predicate succeeds if `Framelist` is the list of frames in which the specified event occurs. In particular, an event consists of the activity type, together with players' names and their roles. The output frame list is a solid set of frames,
2. `frames_to_objects(Frame_ranges, Objects)`. This predicate succeeds just in case `Object` is the set of all objects that occur in a frame occurring in the list, `Frame_ranges`, of frame ranges. An example of such a call could be

   $\leftarrow$ `frames_to_objects`
   ([$<$ `5 10` $><$ `20 30` $>$], `Obj`),

   which asks for all objects that occur in one of the frames $5, \ldots, 9, 20, \ldots, 29$.
3. `frame_lists_to_objects(Frame_range_list, Objects)`: Given a *list of lists* of ranges of frames, this function returns the objects that occur in all of the lists. Thus, for instance, if we call this predicate with the query:

   $\leftarrow$ `frame_lists_to_objects`
   ([[$<$ `2 5` $><$ `24 28` $>$]
   [$<$ `11 14` $><$ `31 39` $>$]], `Objs`)

   then this would find all objects that occur in one of the frames $2, \ldots, 4, 24, \ldots, 27$ and one of the frames $11, \ldots, 13, 31, \ldots, 38$.
4. `object_to_frames(Objname, Frames)`. This predicate finds the frames in which a specified object occurs. For instance, the query

   $\leftarrow$ `object_to_frames`("`Rupert`", `Frames`)

   returns a list, say [$<$ `15 24` $><$ `30 35` $>$], specifying that Rupert appears in frames $15, \ldots, 23$ and $30, \ldots, 34$.
5. `objects_and_events_to_frames(Objectlist, Eventlist, Framelist)`. This predicate takes a list of objects and a list of events (possibly with specified players in specified roles). It succeeds if `Framelist` is the list of frames in which these events occur (with specified characters, if any, in specified roles) and the specified objects occur in these frames as well. As before, the arguments to this predicate are complex types.
6. `frames_to_activities(Frame_range, Activities)`. This predicate is similar to the `frames_to_objects` predicate.
7. `frame_lists_to_activities(Frame_range_lists, Activities)`. Similar This predicate is similar to the `frame_lists_to_objects` predicate.

HERMES. Integrating multiple databases, data structures, and/or software packages is typically accomplished by writing a chunk of program code (called the 'mediator' program; Wiederhold 1992, 1993) that uses the source code of the databases and/or data structures involved to "glue" them together.

HERMES is a system that was developed at the University of Maryland (Subrahmanian et al. 1995) for creating mediators. In HERMES, a mediator is a program, written in a logic-based language (Lu et al. 1994), that we now describe. When a particular set of software packages and/or databases are being integrated, the *mediator author* is charged with the responsibility of writing the mediator, using the mediator development toolkit as his tools.

As the mediator language must be able to extract information from various databases and/or software packages, and, as details of the implementation of these packages may be very hard to fathom/discover, our integration model will use the *existing operations implemented within the packages being integrated* and use the results of these computations (performed within the software packages involved), in further logical manipulations.

AVIS + HERMES. We now quickly describe, through an example, how AVIS and HERMES work together, providing functionality that neither could provide by itself. To see this, consider the query: find all frame sequences (from a movie such as "The Rope") in which an actor, who also acts in a specified movie M, appears. Let us suppose that there is a relation called `portfolio` stored under the INGRES database management system (DBMS) that has the schema `(name,movie)` specifying the movies acted in by a given actor. Furthermore, suppose we have a flat file relation called `cast` (associated with just the single movie "The Rope") with the schema `(name,role)` specifying which actor played which role in "The Rope". This query can be encoded as a special predicate (let us call it `spquery(Actor, Movie, Results)` that succeeds if `Results` is in the appropriate set of frame sequences associated with this query. This can be encoded as a rule in the mediator of the form:

```
spquery(Movie, Results) ←
    in(P, ingres : equal('portfolio',
    movie, Movie)) &
in(Q, flatfile : equal('cast',
    name, P.name)) &
    object_to_frames(Q.role, Results) : 1.
```

The query illustrates some features of the integrated HERMES + AVIS system. In particular:

- The constraint `in(P, relation:equal ('portfolio', movie, Movie ))` performs a select operation on the INGRES relation `portfolio`, producing a set of records related to the given movie.
- The constraint `in(Q, flatfile:equal ('cast', name, P.name ))` performs a select operation on the flat file relation `cast`, identifying all tuples associated with the objects appearing in the given movie.
- Finally, we use AVIS to search our data structures for the appropriate video-frame ids from "The Rope" that contain the character (role) played by this actor. Note that the predicate `object_to_frames(P.role, Results):1`

accesses the AVIS algorithms and AVIS data structures, in the way described within this paper.

When the user wishes to query the video mediator, he interacts with the windown shown in Fig. 3. The subwindow marked "video" in Fig. 3 represents the window in which the query will appear. The subwindow above (which we call the predicate subwindow) contains a list of predefined predicates in the video mediator. The user may select any of these predicates, or define new ones himself in the query subwindow. To see the intended usage and format of a predefined query (e.g., `spquery`) he merely clicks once on that query (e.g. `spquery`) in the predicate window. This brings up the information displayed in the subwindow (called the help subwindow) immediately below the query subwindow. If he clicks again on `spquery`, a string (to be precise, the string `spquery(Movie,Results):1.0` appears in the query subwindow. The user can edit this query (e.g., by replacing "Movie" by the string "Mug Town" in Fig. 3. When he selects the *Search* option, the system executes the query. Figure 4 shows the solution returned by this query. In particular, when video frames are returned as part of the answer, the system merges them into a composite (low-resolution image) answer that the user can view at his/her convenience either as a composite picture, or as a sequence of video. For instance, if the user clicks the answer `comp0.pnm`, then a list of two video frames is shown onscreen. The user may subsequently request a specific image (say `frame2.ras.gz`) – he does this by clicking `frame2.ras.gz` (Fig. 4). To show the images graphically, an MPEG player can be used to play the video, though, in our current implementation, we only bring up still images.

One of the nice advantages of HERMES is that any mediator developed on top of the HERMES platform can have its own application-specific interface. Thus, users who would like to use an SQL-like interface to AVIS can easily do so. It is well known that SQL merely expresses a subset of logic (Ullman 1989); hence writing an SQL front end to AVIS is relatively easy, and one of the students in the HERMES lab, E. Hwang has in fact done so. Figure 5 shows such an example in which the query being posed is: "Find all frame sequences in the movie "The Rope" in which Brendon appears as an object, and the activity taking place is conversation." The list of frame sequences satisfying this requirement is shown in the window labeled "Result of Query Execution."

An obvious thought that comes to mind when implementing systems for querying video data is that, as the majority of the data structures are relational, one could use a relational DBMS to store the content information about the video itself. It is important to note that the combination of HERMES + AVIS outperforms HERMES augmented with a relational representation of the data contained in AVIS. To accomplish this point, we represented all the information related to the movie "The Rope" in a well-known relational DBMS system, INGRES (University version). We then ran experiments based on each of the seven types of queries described earlier in this section. *In all cases that we ran, the HERMES + AVIS combination many times faster than INGRES.* We give the results on four queries that we ran:

On the query `frames_to_objects([10 55],X)`, which says: "Find all objects occurring somewhere in frames 10–55", we obtained the following results:

- INGRES: 52 144.2 ms
- AVIS: 110 ms

We wondered what would happen when the same query was asked with a much narrower range of frames. We asked the query `frames_to_objects([10 12],X)`, which says: "Find all objects occurring somewhere in frames 10–12." We now obtained the following result (averaged over five runs):

- INGRES: 6 903.6 ms
- AVIS: 125 ms

These observations indicate that the AVIS data structures consistently outperform INGRES, and that this difference in performance becomes more marked (in AVIS' favor) as the frame sequences increase in size.

We also asked queries of the form `object_to_frames ("Rupert",Frames)` – this asks for all frame sequences in which Rupert appears. We obtained the following result (averaged over five runs):

- INGRES: 2 213.2 ms
- AVIS: 357.8 ms

An alternative, though similear query, `object_to_frames(''David'',Frames)`, asks for all frame sequences in which Rupert appears with the following results:

- INGRES: 2 053.6 ms
- AVIS: 118.4 ms

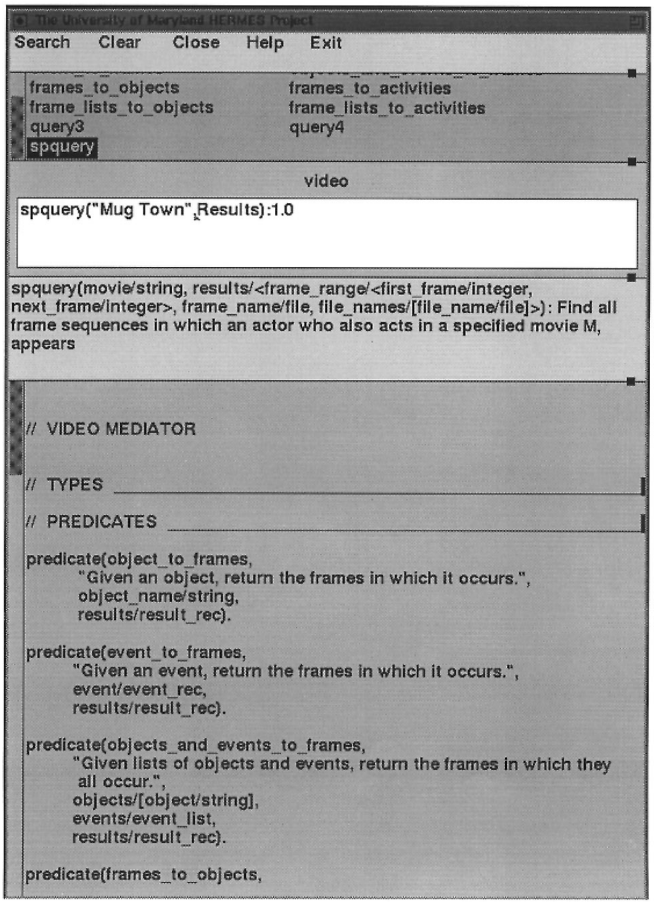
It appears clear from these, as well as from other experiemnts that substantiate the same types of observations reported here, that the AVIS data structures outperform standard relational DBMSs by leaps and bounds when the seven types of operations listed earlier in this section are considered.

8 Related work

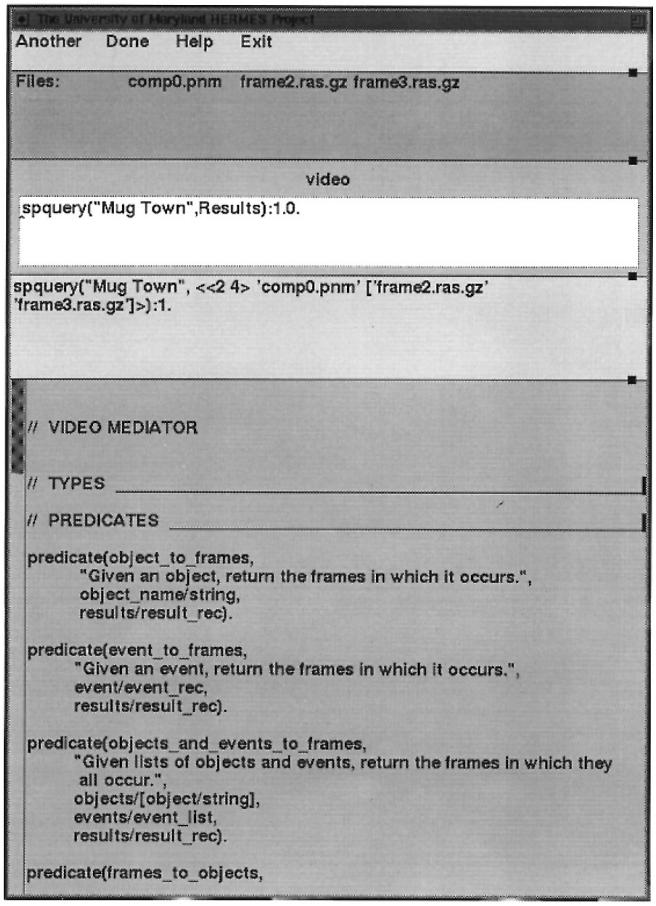
Over the last couple of years, there has been a small, but noticeable, spurt of activity in the area of video databases. The primary aim of this paper is to develop techniques with which video can be organized and queried. Two works that are closely related are those of Gibbs et al. (1994) and Hjelsvold and Midtstraum (1994).

Gibbs et al. (1994) and Gibbs and Tsichritzis (1994) study how stream-based temporal multimedia data can be modeled with object-based methods. However, concepts such as roles and players, the distinction between activities and events, and the integration of such video systems with other traditional database systems are not addressed.

Hjelsvold and Midtstraum (1994) develop a "generic" data model for capturing video content and structure. Their idea is that video should be included as a data type in relational databases, i.e., systems such as PARADOX, INGRES, etc., should be augmented to handle video data. In particular, they study temporal queries – something we do not do



3



4

Fig. 3. Query to HERMES

Fig. 4. Answer to the query to HERMES

in this paper. In contrast, the innovation in our approach is the use of well-studied spatial (rather than temporal) data structures, suitably modified, to query video data. We have used those intuitions to develop techniques for processing queries as well as for *updating* these video data structures – the first algorithms we know of for incorporating updates to video data.

Oomoto and Tanaka (1993) have developed a language called VideoSQL for accessing video data. In contrast to their effort, in this paper we develop indexing structures and query-processing algorithms for querying video data, and use a logical query language to query the data; compare the work on HERMES (Subrahmanian et al. 1995).

Chen and Little (1993; 1995) have studied the organization of video data on disk. Their work is complementary to ours. AVIS develops methods by which we can determine how to answer a query – certain specific type of queries might lead to a necessity to retrieve certain specific frame sequences. Once it is known which frame sequences have the desired content (objects/activities), Chen and Little's (1993, 1995) technique can be used effectively to retrieve those frame sequences from disk.

Other work on video includes work by Davenport et al. (1991) who argue that video should not be segmented at the frame level. This is consistent with our rendition. Segmenting video at the frame level corresponds to a well-known data structure called the *unit* segment tree (Samet 1989), which is just like the segment tree described here except that leaves always must represent unit intervals, i.e., intervals of the form $[i, i + 1)$. In contrast, by using segment trees instead, we allow leaves to have whatever granularity is needed to represent best the content of the video being annotated.

Other works on videl indexing include specific disk-based techniques (Berson et al. 1994; Chen and Little 1993) methods dealing with compression/image processing (Arman et al. 1993), synchronization structures (Iino et al. 1994) and developing ways of handling/presenting continuous data (Rowe and Smith 1992).

There has also been a good deal of work on picture retrieval systems (Gudivada and Raghavan 1993; Gudivada et al. 1994; Niblack et al. 1993). However, these works deal with static images, whereas we have studied ways of organizing temporal sequences of such images (i.e., video).

9 Conclusions

Over the last few years, the ability to deliver movies to homes has increased substantially. The cable TV phenomenon is being enhanced slowly, but steadily, by systems that

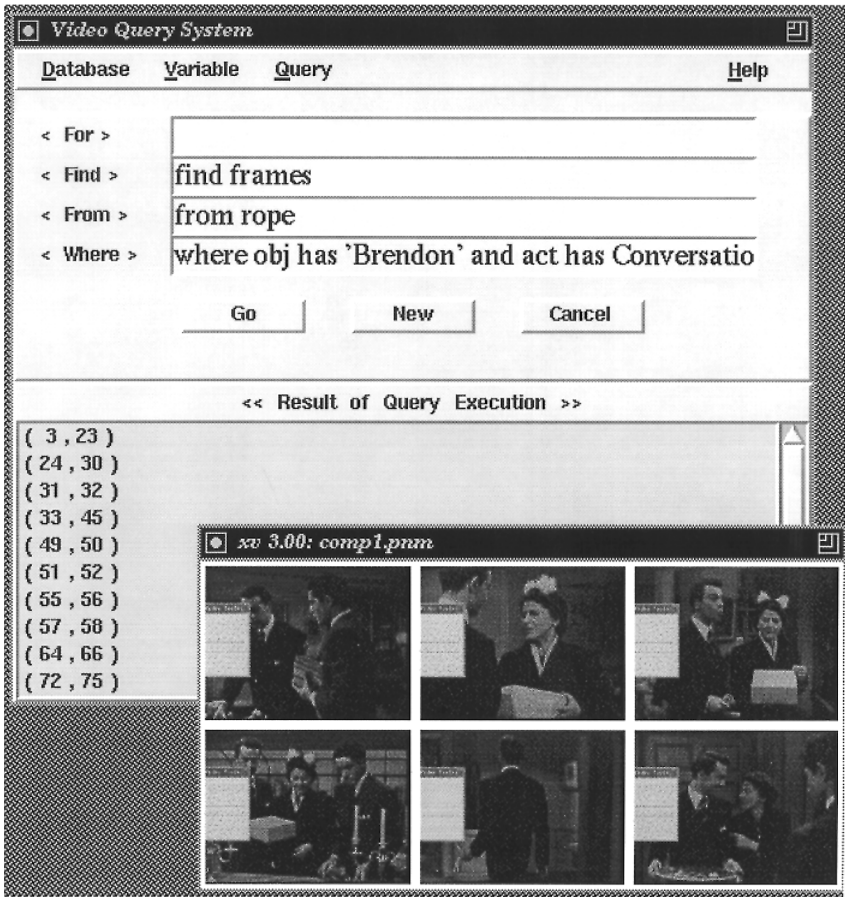


Fig. 5. A simple conjunctive query expressed with an SQL front end to HERMES + AVIS

support the "pay-per-view" paradigm. Over the next few years, with the rapid concomitant evolution of the information superhighway, we expect to see widespread video-on-demand services becoming available. Such services will, in all likelihood, not just encompass commercial movies from the entertainment industry, but are likely to be broad and include movies about travel destinations, movies about educational items, movies and advertising material on consumer products, training manuals, etc.

Once an electronically accessible video library exists (and some such libraries exist even today, e.g., at the National Archives in College Park, Md.), the need to access these databases "by content" assumes great importance. The primary aim of this paper is to develop techniques by which the content of these movies can be stored online so as to facilitate such accesses.

To accomplish this, we have presented a succinct theoretical description of video databases. We have shown how such video databases can be stored electronically, and furthermore, we have designed polynomial time, query processing algorithms that traverse these data structures. In addition, as the content of these video databases can be updated from time to time, we have developed methods to implement such updates. Finally, we have developed an implemented AVIS, which implements the various video-based queries described here.

Acknowledgements. This research was supported by the Army Research Office under grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under grant F49620-934-1-0065, by ARPA/Rome Labs contrat No. F30602-93-C-0241 (Order No. A716), and by an National Science Foundation (NSF) Young Investigator award IRI-93-57756.

References

Arman F, Hsu A, chiu M (1993) Image processing on compressed data for large video databases. First ACM international Conference on Multimedia, Anaheim, CA, pp 267–272

Berson S, Ghandeharizadeh S, Muntz R, Ju X (1994) Staggered striping in multimedia information systems. Proceedings of the 1994 ACM SIGMOD Conference on Management of Data, Minneapolis, MN, pp 79–90

Chen HJ, Little T (1993) Physical Storage Organizations for time-dependent multimedia data. Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference, Evanston, IL, pp 19–34

Chen HJ, Little T (1995) A storage and retrieval technique for the provision of scalable MPEG video. To appear, J Parallel Distributed Comput

Davenport G, Smith TA, Pincenver N (1991) Cinematic prinmitives for multimeida. IEEE Comput Graph Appl 11:67–74

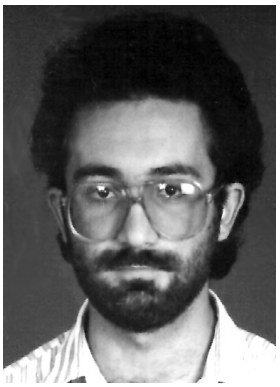
Gibbs S, Tsichritzis D (1994) Multimedia programming: objects, environments and frameworks. ACM Press/Addison Wesley

Gibbs S, Breiteneder C, Tsichritzis D (1994) Data modeling of time-based media. Proceedings of the 1994 ACM SIGMOD Conference on Management of Data, Minneapolis, MN, pp 91–102

Gudivada VN, Raghavan VV (1993) Design and evaluation of algorithms for image retrieval by spatial similarity. ACM Trans Information System

Gudivada N, Raghavan VV, Vanapipat K (1994) A unified approach to data modeling and retrieval for a class of image database applications.

In: Jajodia S, Subrahmanian VS (eds) Multimedia database esystems. Springer, Berlin Heidelberg New York, pp 37–78, 1995
Hjelsvold R, Midtstraum R (1994) Modeling and querying video data. Processings of the 1994 International Conference on Very Large Databases, Santiago, Chile, Morgan Kaufman, pp 686–694
Iino M, Day YF, Ghafoor A (1994) An object-oriented model for spatio-temporal synchronization of multimedia information. Proceedings of the 1994 International Conference on Multimedia Computing and Systems, Boston, MA, IEEE Press, pp 110–119
Lu J, Nerode A, Subrahmanian VS (1994) Hybrid knowledge bases. Technical Report CS-TR-3037, University of Maryland, IEEE Trans Knowledge Data Eng
Marcus S, Subrahmanian VS (1994) Foundations of Multimedia Database Systems, Journal of the ACM
Niblack N (1993) The QBIC project: querying images by content using color, texture and shape. IBM Research Report, Almaden, CA
Oomoto E, Tanaka K (1993) OVID: design and implementation of a video-object database system. IEE Trans Knowledge Data Eng 5:629–643
Rowe L, Smith BC (1992) A continuous media player. Proceedings of the 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video, San Diego, CA, pp 237–249
Samet H (1989) The design and analysis of spatial data structures. Addison-Wesley, Reading, MA
Subrahmanian VS (1995) HERMES: a heterogeneous reasoning and mediator system
Ullman JD (1989) Principles of database and knowledge-base systems. Computer Science Press
Wiederhold G (1992) Mediators in the architecture of future information systems. IEEE Computer, pp 38–49
Wiederhold G (1993) Intelligent integration of information. Proceedings of the ACM Conference on Management of Data, Washington, D.C., pp 434–437

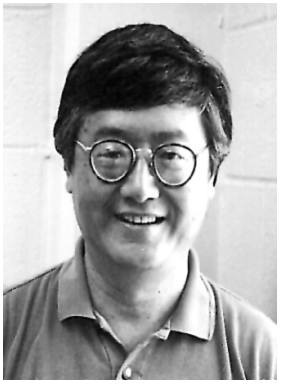


KUTLUHAN EROL is a Senior Scientist at Intelligent Automation Inc. He got his B.S. degree in computer science from Bilkent university in Ankara in 1990. He pursued his studies at University of Maryland, where he obtained his Ph.D. in computer science in 1995. His research interests include planning, scheduling, factory automation, multimedia, agent-based architectures, and complexity theory.

V.S. SUBRAHMANIAN received his PhD in Computer Science from Syracuse University in 1989 and currently is an Associate Professor in the Computer Science Department at the University of Maryland, College Park. He received the prestigious NSF Young Investigator Award in 1993. He has worked extensively in the fields of nonmonotonic reasoning, reasoning with inconsistency and uncertainty, databases, and in integrating multiples forms of data, software and reasoning paradigms. He has published over 65 papers in prestigious journals and conferences. He has also given invited talks and served on invited panels at several leading conferences.

KASIM SELCUK CANDAN is a graduate student in the Computer Science Department of the University of Maryland at College Park. He is working in the HERMES project. He received his B.S. degree in computer science from Bilkent University in Turkey in 1993. His research interests include multimedia databases and multimedia collaboration.

SIBEL ADALI is a PhD candidate at the University of Maryland where she is currently working on the development of the Hermes System (http://www.cs.umd.edu/projects/hermes) for integrating multiple forms of data, software and reasoning paradigms. Her research interests include optimization in heterogeneous information systems, interoperability in database systems, reasoning with uncertainty and multimedia information systems. She received her BS degree in Computer Science at Bilkent University, Turkey in 1991 and her MS degree in Computer Science at the University of Maryland in 1994.



SU-SHING CHEN is Visiting Senior Scientist at the Institute for Systems Research, University of Maryland, College Park, and Professor of Computer Science at the University of North Carolina at Charlotte. He received his PhD from the University of Maryland in 1970. His research interests include image processing, visualization, multimedia computing, and digital libraries.