Similarity Based Retrieval of Videos

A. Prasad Sistla Clement Yu * Raghu Venkatasubrahmanian

Abstract

In this paper, we propose a language, called Hierarchical Temporal Logic (HTL), for specifying queries on video databases. The language is based on the hierarchical as well as temporal nature of video data. We give similarity based semantics for the logic, and give efficient methods for computing similarity values for subclasses of HTL formulas. Experimental results, indicating the effectiveness of our methods, are presented.

1. Introduction

We are currently witnessing an explosion of interest in multimedia technology. Consequently, pictorial and video databases will become central components of many future applications. Access to such databases will be facilitated by a query processing mechanism that retrieves pictures based on user queries. In our earlier works [27, 25, 26, 2], we had presented methods for retrieving pictures based on their contents. These works employ similarity based methods for retrieving pictures that use indices on spatial as well as non-spatial relationships of objects. In this paper we present our system for similarity based retrieval of videos. Our system consists of methods for similarity based retrieval that are built on top of an existing picture retrieval system (described in [27, 25, 2]).

We assume that there is a database containing the actual videos, and another database that contains the meta-data describing the contents of the various videos. We use a hierarchical model for representing the meta-data. In this model, a single video is arranged into various levels. Each level consists of a temporally ordered sequence of video segments which is a decomposition of video segments at the next higher level. At the top level we simply have a single video segment representing the whole video. The meta-data associated with this video segment may give the name of the video, the actors and other relevant information. At the next level this video may be decomposed into a sequence of sub-plots, and each sub-plot may be further decomposed into a sequence of scenes at a lower level; at a still lower level, each scene may be decomposed into a sequence of shots. At the lowest level each shot is a sequence of frames. We assume that there is meta-data associated with each video segment at each level in the above hierarchy describing the contents of the video segment. This meta-data contains information about the objects in the video segments, their properties and the relationships among them.

In order to specify queries, we introduce a Hierarchical Temporal Language (HTL). This language uses the classical temporal operators to specify properties of video sequences (i.e. the temporal properties); it also has certain level modal operators that allows us to specify properties of the sequences at different levels in the above hierarchy. At the atomic level, the language allows specification of properties on the meta-data associated with a single video segment (such as a shot or a frame; in practice a key frame can be extracted from a shot and meta-data is associated with the key frame). More complex specifications are formed from the atomic specifications by combining them using temporal operators as well as level modal operators together with boolean connectives. In order to be accessible to a naive user, our query language is built on top of an icon based picture retrieval system [2, 27, 25].

It is generally accepted that retrieval from multimedia databases should be similarity based rather than based on exact matching. This is partly due to the fact that the user may not know exactly what he/she wants. Even otherwise, the database may not have video segments that exactly match with the user's query, or the meta-data used may not be completely accurate due to the limitations of the image analysis algorithms. Thus, one would like to retrieve video segments that closely/approximately match with the user's query. With this as the main motive, we provide a similarity based semantics for the formulas of HTL. Under this semantics, for each video segment and each HTL formula, we define a similarity value that denotes how closely the video segment satisfies the formula. The semantics that we give are quite intuitive. Under our similarity based retrieval, the k top video segments that have the highest similarity values with respect to the user query will be retrieved; here, k may be a parameter specified by the user.

After presenting the similarity based semantics, we give efficient algorithms that compute the similarity values of the relevant video segments with respect to the given user query. For each atomic specification s in the user query, these algorithms retrieve a list of similarity values of the relevant video segments denoting how closely the atomic specification s is satisfied by the video segments (retrieval of such lists can be done using similarity based picture re-

*The address of first two authors: Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607. The research of Clement Yu is partly supported by NSF grant NSFIRI-9509253.

trieval systems employing indices on the meta-data; see [27, 25, 2]). The retrieved lists are combined together in an inductive manner to obtain a list of similarity values of relevant video segments with respect to the main user query. After this the top k video segments, having the highest similarity values, are presented to the user. We classify four classes of HTL formula— type (1), type (2), conjunctive and extended conjunctive formulas (each class in this list is a subclass of the next one). For each of these classes of formulas, we provide efficient algorithms for computing similarity values. Some of these algorithms are implemented on top of existing similarity based picture retrieval system. We present some experimental results on real data as well as on some artificially generated data. Initial results indicate that our algorithms for retrieving video segments are fairly efficient.

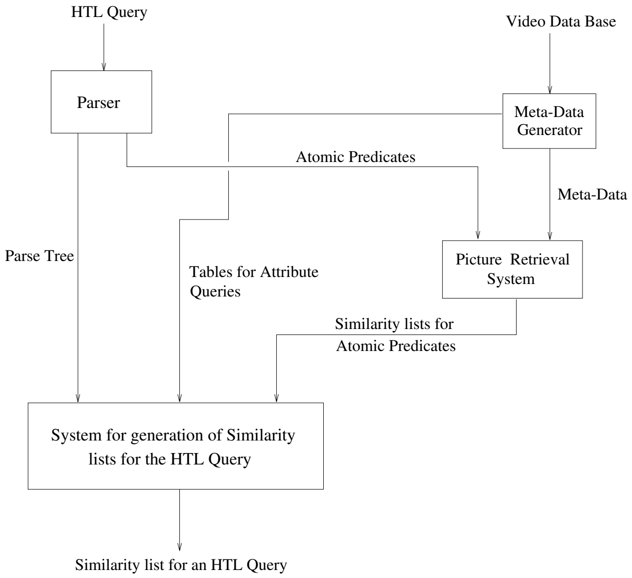

Figure 1. Architecture of the Video Retrieval System

The overall architecture of the video retrieval system is shown in figure *1*. It consists of a parser that parses the video query and identifies the atomic predicates and generates the parse tree. The video analyzer consists of a system that generates the meta-data; this may itself consist of systems for segmentation, editing [11, 21, 23] of video data as well as algorithms for analysis of the video. The picture retrieval system generates the similarity lists corresponding to the atomic predicates using the meta-data (here we used the system described in [27, 2]; however any compatible system can be employed). The component similarity list generator generates the similarity list corresponding to the main HTL video query; this is what we present in this paper.

In summary, the main contributions of the paper are as follows.

- A hierarchical model for video databases and a Hierarchical Temporal Language for specifying queries on the video databases together with a similarity based semantics for the language.
- Efficient algorithms for various important subclasses of formulas of HTL. These algorithms are built on top of an existing picture retrieval system.
- Experimental results denoting the effectiveness of our approach.

There has been some earlier work on modeling video data and specifying video queries [4, 6, 7, 10, 14, 12, 15, 30, 3, 19]. Many of these works use some type of temporal relationships in their query languages. However, all of them either use exact matching for retrieval purposes (e.g. [30, 12]), or employ keywords or annotated textual descriptions (e.g. [4]). For example, the work presented in [30] introduces a video algebra for playing and querying video and is also based on hierarchical organization of the video; in their scheme retrieval is based on exact match, and also their query language does not permit comparison of attribute values in different video segments of a video. Efficient querying of pictures by content employing multidimensional indexing has been done in QBIC and other projects [17, 8]; these static image querying systems have been extended using key frames to videos also in [7, 9]. The above querying systems are similarity based but they do not employ temporal operators for querying purposes. In contrast, our video retrieval system is similarity based; our query language permits comparison of attribute values in different video segments; it employs temporal operators, and modal operators corresponding to the different levels in the video hierarchy. Furthermore, we give efficient retrieval algorithms together with accurate complexity analysis which is lacking in earlier works.

Our paper is organized as follows. Section 2 describes the model and the language HTL. Section 3 gives the algorithms used in the similarity based retrieval. Section 4 contains the experimental results. Section 5 contains the conclusions.

2. The Model and the Specification Language

2.1 Model

Videos can be conceptualized (or arranged) hierarchically as follows. At the top level, a video can be classified according to its type. For example, it can be a western movie or a news broadcast or a military operation. At the next level, a video may be decomposed into a sequence of sub-plots. For example, a video on the Gulf war may be composed of a sub-plot denoting bombing of the Iraqi positions, followed by a sub-plot denoting the involvement of allied armies against the Iraq army, followed by the sub-plot denoting the surrender of the Iraq troops. At a lower level, each sub-plot may be divided into a sequence of scenes. For example, the sub-plot denoting the bombing of Iraq may be divided into a scene involving the bombing the command and control centers followed by a scene denoting the bombing of airfields etc. At a still lower level, each scene may be divided into a sequence of shots. For example, the bombing of the command and control centers starts with a shot denoting the take off of the fighter planes and the bombers, followed by a shot where the bombs were dropped and the targets were destroyed, followed by the return of the planes. At the lowest level each shot is a sequence of pictures.

We define a video segment to be any contiguous part of the video. The entire video, any plot, sub-plot, scene, shot and picture are all video segments. From the above discussion, we see that the above hierarchy can be represented as a tree with the nodes of the tree denoting various video segments. The leaves in the tree denote actual pictures. We also assume that all the leaves in the tree lie at the same level, i.e. at the same depth from the root. We assume that there is meta-data associated with the various video segments in the above hierarchy. For example, the meta-data associated with the video at the top level, may indicate that it is a western movie and the associated relevant information such as the title, the actors, the length of the movie etc. The meta-data associated with a sub-plot may indicate it to be an attack by bandits on a village. The meta-data of a scene may indicate the approach of bandits on horses. The meta-data of a shot may indicate it to be an event where John Wayne shoots a bandit. The meta-data associated with a picture may denote the presence of John Wayne holding a gun. We use an extended E-R model for the meta-data (see [27, 25, 2]).

A video query is specified by providing information about the properties that need to be satisfied by the various video segments at different levels of the above hierarchy. If the information provided in the query pertains to the upper levels only, then the user is interested in browsing. For example, the information provided by a browsing query may indicate western movies starring John Wayne and nothing else. The user query may also provide information by specifying temporal properties on the sequence of video segments at any of the levels. For example, the user may query for all videos starting with a scene depicting John Wayne shooting a bandit, later followed by a scene where John Wayne unites with his people.

2.2 Syntax

We propose a formal query language called *Hierarchical Temporal Logic*(HTL) for specification of user queries on video databases. The actual user query, for a naive user, is specified in a user friendly graphical language and is translated into this formal language. The language is based on the above hierarchical arrangement of the meta-data. We associate a level number with each level in the above hierarchy, with the root being level 1. Assume that there are n levels in the above hierarchy; with level n containing the meta-data associated with individual frames. There may also be names associated with the levels; for example, scene level and frame level, denote the levels where all the video segments are scenes, frames respectively.

Now, we describe the syntax of HTL. The formulas of HTL are formed using predicates on the meta-data for the individual video segments, the special unary predicate $present$, two types of variables called object variables and attribute variables, various point based temporal operators such as **eventually**, **until**, **next** (see [16]), the standard propositional connectives such as $\wedge$ ("and"), and $\neg$ ("not"), first order existential quantifier, the assignment operator $\leftarrow$ (also called the *freeze* quantifier) and different modal operators corresponding to various levels such as **at-next-level, at-level-i, at-shot-level, at-frame-level, at-scene-level** (called level modal operators). We assume that there is a universal set of object ids and each object in a picture is assigned an object id such that the same object in different pictures is given the same id, and different objects have different ids. (Using current technology, it is possible to track an object; that is, once an object is identified in a frame of a scene, it is easy to track it in subsequent frames until it disappears from the scene).

The unary predicate $present(x)$ denotes if object x is present in a video segment. The assignment operator $\leftarrow$ (also called the freeze quantifier) allows us to capture the value of an attribute in a video segment and compare it with attribute values in later video segments; this operator can be considered as a restricted form of first order quantifier. The temporal operators allow us to specify properties on a sequence of video segments at the same level, while the level modal operators allow us to specify properties of sequences of video segments at different levels.

In order to define the syntax of HTL formulas, we define the syntax of expressions (i.e. terms) as in the case of first order predicate calculus. Now, HTL formulas are defined recursively as follows. If x is an object variable then $present(x)$ is a HTL formula; if $e_1, e_2, ..., e_k$ are expressions and P is a k-ary predicate symbol of arity k and is of appropriate type, then $P(e_1, ..., e_k)$ is a HTL formula; if f,g are HTL formulas then $\mathbf{next}\ (f)$, $f\ \mathbf{until}\ g$, $\mathbf{eventually}\ f$, $f \wedge g$, $\neg f$, $\exists x(f)$, $[y \leftarrow q](f)$ where x, y are object and attribute variables respectively, and q is a function returning an attribute value of an object in a video segment, **at-next-level**(f), **at-level-i** (f) for $i = 1, ...n$, **at-scene-level**(f), **at-shot-level**(f), **at-frame-level**(f) are all HTL formulas. The logic HTL is an extension of Future Temporal Logic (FTL) that was defined in [24].

Let f be a HTL formula. We say that a variable x appearing in f is bound if all the occurrences of x appear in the scope of an existential or freeze quantifier over x. A variable x is said to be free in f if it is not a bound variable in f. An evaluation ρ for f is a function that assigns values to the variables appearing free in f. We say that a formula is *non-temporal* formula if it has no temporal operators and has no level model operators in it. Note that a non-temporal formula asserts a property on the meta-data associated with a single video segment.

2.3 Semantics

Consider any node, i.e. video segment, in the hierarchical tree of a video v. All the descendants of this node at some level l can be arranged in a sequence from left to right; we call such sequences as *proper* sequences of v. For a HTL formula f, an evaluation ρ for f, and for any proper sequence s of v and any video segment u in the sequence s, we inductively define when f is satisfied at u. For a formula f that does not contain any temporal operators, or level modal operators or quantifiers, its satisfaction only depends on the video segment u and is independent of the rest of the sequence; in this case the satisfaction is defined as in the case of first order logic. The semantics of the different temporal operators are defined as usual. For example, if $f = g$ **until** h , then f is satisfied at u in s with respect to ρ if there exists a video segment u' which is same as u or which appears after u in s such that h is satisfied at u' in s with respect to ρ and g is satisfied at all video segments between u and u' in s. Note that if h is satisfied at u then f is satisfied at u irrespective of whether g is satisfied at u or not. If $f =$ **next** g then f is satisfied at u in s with respect to ρ if g is satisfied at the video segment immediately after u. If f is **at-next-level**(g) then f is satisfied at u in s if g is satisfied at the first video segment of the sequence s' consisting of all children of u; if u has no children then f is not satisfied at u. Similarly, the semantics of other level modal operators are defined by considering the sequence of descendents of u at that level. The formula $[x \leftarrow q]\ g$ is satisfied with respect to ρ at u if g is satisfied at u with respect to ρ' where $\rho'(x)$ is the value of the attribute q in u, and for all other variables y, $\rho'(y)$ is same as $\rho(y)$. The semantics of a formula of the form $\exists\ x\ g$ is defined similarly. We say that a formula f is satisfied by a video v if it is satisfied at the root in the sequence consisting of only the root (i.e. a sequence of one element).

2.4 Examples

The following formula when asserted at the shot level specifies that the sequence starts with a shot in which some planes (one or more) are on the ground (denoted by the non-temporal predicate M_1), followed immediately by a sequence of shots in which some planes are in the air (denoted by the predicate M_2) until a shot in which a plane was shot down (denoted by M_3). This formula can be asserted at the shot level in the video hierarchy by prefixing it with the level modal operator **at-shot-level**.

(A) $\quad M_1 \wedge \textbf{next}\ (M_2\ \textbf{until}\ M_3)$

Now, as another example, a video scene in which John Wayne shoots a bandit can be specified by requiring the presence of three frames where the first frame has both John Wayne and the bandit holding guns, the second frame has John Wayne firing at the same bandit, and the third frame has that bandit on the floor. The following formula f when asserted at the frame level specifies this property.

(B)
$\exists x \exists y\ P_1(x,y) \wedge$ **eventually**($P_2(x,y) \wedge$ **eventually** $P_3(y)$)
where $P_1(x,y)$ is the subformula
$(present(x) \wedge present(y) \wedge person(x) \wedge person(y) \wedge name(x) = `JohnWayne' \wedge$
$type(y) = `bandit' \wedge (x$ holds a gun $) \wedge (y$ holds a gun $))$,
$P_2(x,y)$ is the subformula $present(x) \wedge present(y) \wedge (x$ fires at $y)$,
and $P_3(y)$ is the subformula $present(y) \wedge (y$ on the floor$)$.

In the formula f given above, "x holds a gun", "y holds a gun", "x fires at y" denote binary predicates. Now, the formula $type = western \wedge$ **at-frame-level**(f) where f is the above formula asserts that the video is a western movie which has a sequence of frames in which John Wayne shoots the bandits.

The following formula, asserted at the frame level, specifies that the video starts with a picture containing an airplane followed by another picture in which the same plane appears at a higher altitude.

(C) $\exists x(Q_1(x) \wedge [h \leftarrow height(x)] \textbf{eventually} Q_2(x,h))$
where $Q_1(x) = (present(x) \wedge type(x) = `airplane`)$
and $Q_2(x,h) = (present(x) \wedge height(x) > h)$

In the formula (C), $Q_1(x)$ asserts that x is an airplane and is present in the picture, $Q_2(x,h)$ asserts that x is present in the picture and that its height is greater than h. In (C), we use the assignment operator (using variable h) to capture the height of x in the first picture and compare it with the height of x in the later picture.

2.5 Similarity based semantics

In this section we define the similarity based semantics of HTL formulas. We consider four subclasses of HTL formulas. First, we define two subclasses called *conjunctive* formulas and *extended conjunctive* formulas. We say that a HTL formula is a conjunctive formula if it has no negations, no level modal operators and all the variables in the formula are bound by an existential quantifier or by an assignment operator, and furthermore each existential quantifier either appears at the beginning of the formula or has no temporal operators appearing in its scope. Note that a conjunctive formula f is of the form $\exists x_1, ..., \exists x_n g$ where the formula g can syntactically be visualized to have been formed from a set of non-temporal formulas connected by temporal operators, the assignment operator and the conjunction operator; note that any existential quantifier appearing in g is part of the non-temporal formulas. All the formulas given above by (A), (B) and (C) are conjunctive formulas. We believe that most queries of interest can be expressed as conjunctive formulas. An *extended conjunctive formula* is a formula satisfying all the above conditions except that it can also have level modal operators.

Let f be an extended conjunctive formula and ρ be an evaluation for f. Also, let v be a video, s be a proper sequence of v and u be a video segment in s. Now we define a similarity value that denotes how closely the formula f is satisfied at u in s with respect to ρ. The similarity value is given by a pair of positive numbers (a, m) where $a \leq m$; here a denotes the *actual similarity* value and m denotes the maximum possible value; for an exact match (i.e. if u in s exactly satisfies f) a and m will be equal; we assume that the maximum m is only a function of f (i.e. it is independent of u, s etc.). Intuitively, we can consider the similarity value to be $\frac{a}{m}$ which we call as the *fractional similarity* value.

- If f has no temporal operators or has no level modal operators then the satisfaction of f with respect to ρ only depends on the meta data associated with the video segment u and is independent of the rest of s and v. In this case, the similarity value is defined exactly as in the case of similarity values for pictorial queries as given in [25, 27].

- If $f = g \wedge h$ and the similarity values of g and h at u in s with respect to ρ are given by (a_1, m_1) and (a_2, m_2), then the similarity value of f at u in s with respect to ρ is given by $(a_1 + a_2, m_1 + m_2)$. It is to be noted that even if one of a_1 and a_2 is zero, the actual similarity value of f which is $a_1 + a_2$ may be non-zero. That is, even if one of g and h is not at all satisfied, we still may consider f to be partially satisfied. Note that this definition of the similarity value satisfies all the desired properties (a),(b) and (c) discussed earlier.

- If $f = \textbf{next } g$ then the similarity value of f at u in s is same as the similarity value of g at u' where u' is the video segment immediately next to u in s; if u is the last video segment in s then the similarity value of f is $(0, m)$ where m is the maximum possible similarity value for g.

- If $f = g \textbf{ until } h$ then we define the similarity value as follows. Intuitively, we consider f to be partially satisfied at u if there exists a later video segment u'' where h is partially satisfied and at all video segments between u'' and u, g is satisfied with a minimum threshold value; we take the similarity value of f to be the maximum of the similarity values of h at all such video segments u''. Note that this definition intuitively corresponds to the exact semantics of the "until" operator.

- If $f = [x \leftarrow q]\ g$ then the similarity value of f at u in s with respect to ρ is the same as the similarity value of g at u in s with respect to ρ', where $\rho'(x)$ is the value of the attribute q in u, and for all other variables y, $\rho'(y) = \rho(y)$.

- If f = **at-next-level** (g) then the similarity value of f at u is same as the similarity value of g at the first video segment in the sequence formed by the children of u. In case u has no children then the actual similarity value of f at u is defined to be zero. The definitions for other level model operators are defined by considering the descendents of u at the appropriate level.

- If $f = \exists x_1, ..., x_k\ g(x_1, ..., x_k)$ where $x_1, ..., x_k$ are all the free object variables appearing in g, then the similarity value of f at u is (a, m) where a is the maximum of the set of actual similarity values of g at u with respect to different evaluations of g, i.e. $a = max\{$actual similarity of g at u with respect to $\rho : \rho$ an evaluation for $g\}$, and m is the maximum similarity value of g; note that in our scheme the maximum similarity value of g at u with respect to ρ depends only on g and is independent of u and ρ.

3. Algorithms for Similarity Based Retrieval

In the previous section we had introduced a formal query language for specifying queries on video databases and defined subclasses of formulas. In this section, we present methods for computing similarity values of videos with respect to specifications given by the different subclasses of formulas.

We assume that each video has only two levels, the root node and its children where all the children are of the same type, i.e. they are all frames, or they are all shots, or they are all scenes. We assume that the given conjunctive temporal formula is asserted on the sequence consisting of the children of the root node. We slightly abuse notation and say that the video is simply a sequence of video segments. We will later show how our algorithms can be extended to the case where the video has more than two levels and the given HTL formula contains level modal operators.

First, we define two subclasses of conjunctive formulas called type (1) and type (2) formulas. A conjunctive formula is a type (1) formula if it has no assignment operators and if it has no temporal operators appearing in the scope of the existential quantifiers. Essentially, a type (1) formula can be visualized as a set of non-temporal formulas connected together by the temporal operators and the conjunction operator; in this case, any existential quantifiers in the formula will be considered as part of the non-temporal formulas. A type (2) formula is simply any conjunctive formula not containing assignment operators. Clearly, type (1) formulas form a subclass of a type (2) formulas. The formulas (A) and (B), given earlier, are type (1) and type (2) formulas respectively, while the conjunctive formula given by (C) is neither a type (1) nor a type(2) formula.

3.1 Algorithm for type (1) formulas

We assume that each video is a sequence of video segments and that these segments are numbered sequentially starting from 1. For the present, we assume that we only have a single video; multiple videos can be handled by using two numbers one of which gives the video id and the other gives the id of the video segment within the video. Given a formula f of type (1), the algorithm generates a list of entries of the following form $([beg\_id, end\_id], (act\_sim, max\_sim))$ $[beg\_id, end\_id]$ denotes an interval (i.e. a sequence of video segments beginning with beg_id and ending with end_id), act_sim and max_sim denote the actual and maximum similarity values (note that this list is a table/relation with four attributes). Each such entry indicates that the formula f has the fractional similarity value $\frac{act\_sim}{max\_sim}$ at all the video segments with ids beg_id, end_id and those that lie between them. For any id not belonging to any interval on the the list, the actual similarity at that video segment is zero, i.e. only ids with non-zero similarity value appear on the list. It is to be noted that max_sim is the same for all entries and its value only depends on f.

We call a list of the above type as a *similarity list* or a *similarity table*. For a similarity list L, we define $length(L)$ to be the number of entries in L.

Now we present the algorithm for type (1) formulas. Recall that any type (1) formula is formed from non-temporal formulas using the conjunction (i.e. $\wedge$) and the temporal operators. Our algorithm works inductively on the structure of the formula f as given in the following cases.

f is a non-temporal formula: In this case, the similarity value of f at a video segment u in a proper sequence s only depends on u and is independent of the rest of s. For this reason, we can process f as a query on a picture or on a single video segment. Using the methods given in [27] and employing indices, we simply retrieve a list of entries that give the similarity value of f at different video segments within the video. It is to be noted that any video segment with zero similarity value for f will not appear on this list.

$\mathbf{f} = \mathbf{g} \wedge \mathbf{h}$: Let L_1, L_2 be the lists giving the similarity values for g and h respectively. It is to be noted that the intervals appearing on L_1 are disjoint, i.e. non-overlapping. Similarly the intervals in L_2 are disjoint. If a video segment appears in both the lists then its actual similarity value is the sum of the its actual similarity values on the two lists; if it appears only in one list then its actual similarity is same as the value in that list. The algorithm first sorts the two lists L_1 and L_2 on the beginning ids of the intervals, and employs a modified merge of the two sorted lists to compute the similarity list for f. If the two lists are already sorted then the complexity of the algorithm is $O(length(L_1) + length(L_2))$, otherwise it will be $O(length(L_1)\log(length(L_1)) + length(L_2)\log(length(L_2)))$ including the complexity to sort the input lists.

$f = \mathbf{next}\, g$**:** Observe that if g has a similarity value of (a, m) during the interval $[u, v]$, then the similarity value of $\mathbf{next}\, g$ is (a, m) during the interval $[u-1, v-1]$. Now, the similarity list for f is obtained by replacing the interval $[u, v]$ in each entry of the input list by the interval $[u-1, v-1]$.

$f = g\, \mathbf{until}\, h$**:** Let L_1 and L_2 be the lists corresponding to g and h respectively. We assume that the lists are sorted in increasing values of the beginning ids of each entry. Note that from the definition of the similarity for f, it is obvious that, as far as g is concerned, we are only interested in whether the fractional similarity of g is above or below the threshold value at each video segment; the actual similarity values of g are not used in the computation. For this reason, from L_1 we delete all those entries whose fractional similarity values are below the threshold value. After this, we combine all consecutive entries on L_1 whose intervals are adjacent (i.e. the interval of a succeeding entry begins with an id which is immediately after the end of the previous entry) into a single entry (note that we don't need their actual similarity values). After this processing, it should be clear that there will be a gap between the intervals of any two successive entries on L_1.

The algorithm is based on the following property which follows from the definition of the similarity value for f. Consider any entry I in L_1 and an id number i in the interval of I. The actual similarity value of f at the video segment i is the maximum of all the actual similarity values associated with the entries in L_2 whose intervals intersect with that of I at some point greater than or equal to i. Now consider an interval J in L_2 and an id number j in the interval of J. If there is no entry in L_1 whose interval contains j then the similarity value of f at j is the same as the similarity value contained in the entry J.

The algorithm is similar to a merge algorithm and it processes the two lists are processed in the order of descending values of the end ids of their entries (this is same as the order of descending values of the beginning ids of the entries); to do this, the algorithm reads the lists backwards starting from the last entries. The details of the algorithm will be given in the full paper.

Example: The following example illustrates the operation of the algorithm. Assume that L_1 has the following two entries ([25 100], (*,*)), ([200 250], (*, *)) after discarding entries with similarity values below the threshold. The similarity values in the L_1 list are given as *s since their values are not used any more. Assume that L_2 has the three entries ([10 50], (10,20)); ([55 60], (15 20)); ([90 110], (12 20)); ([125 175], (10 20)). The output of the algorithm will have the following entries— ([10 24], (10 20)); ([25 60], (15 20)); ([61 110], (12 20)); ([125 175], (10 20)). This example is pictorially depicted in figure 2.

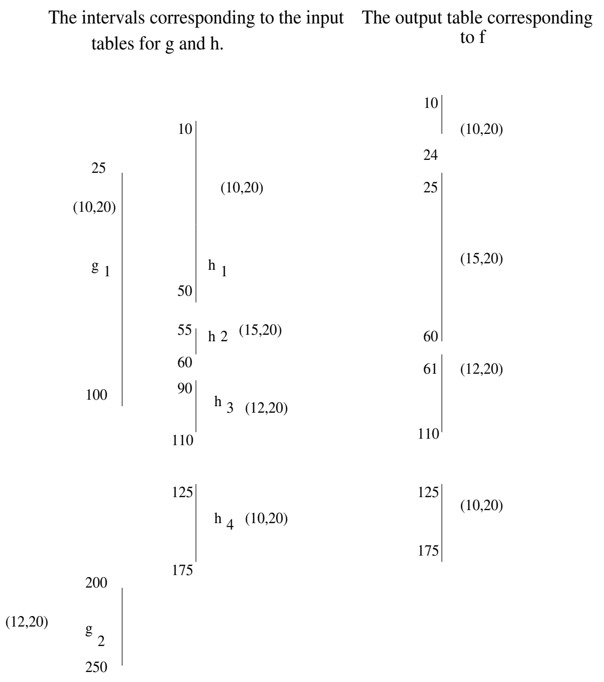

Figure 2. Example of the algorithm for until

The complexity of the algorithm is $O(length(L_1) + length(L_2))$.

Let f be an arbitrary type (1) formula of length p, and l be the sum of the lengths of the similarity lists for all the atomic predicates appearing in f. Then it can be shown that the over all complexity of the above algorithm when applied to f is $O(lp)$.

3.2 Algorithm for type (2) formulas

In this subsection we describe the algorithm for computing similarity lists for type (2) formulas. Now consider a formula f given by $\exists x_1 \exists x_2 ... \exists x_n\ g(x_1, ..., x_n)$. The algorithm for f has two parts. In the first part, we compute a *similarity table* T for g and in the second part we compute the similarity list for f from the table T.

During the first part, for each subformula h of f, we inductively compute the similarity table for h. Assume that the subformula h has k object variables appearing free in it. The similarity table for h will have $k+1$ columns. The first k column names will be the names of the free variables appearing in h and the $(k+1)st$ column will be a similarity list, which is a list of pairs giving the similarity values in different intervals of video segments. In each row, the values of the first k columns gives an evaluation ρ for the formula, and the value of the last column is a similarity list denoting the similarity values of h at various video segments with respect to the evaluation ρ.

Now, the similarity table for a subformula h of f is computed inductively as follows. If h is non-temporal then the table for h is computed using the approach given in [27]. Now, consider the case when $h = h_1 \wedge h_2$ or $h = h_1$ **until** h_2. Assume that the number of free variables in h_1, h_2 and h is given by k_1, k_2 and k respectively. Assume that T_1 and T_2 are the tables computed for h_1 and h_2 respectively. We compute the table T as follows. The values in the first k columns of T are obtained by simply making a join of the first k_1 columns of T_1 with the first k_2 columns of T_2 where the join condition is the equality condition for the common column (i.e. variable) names. If rows r_1 and r_2 are joined to obtain the first k columns of a row r in T, then the $(k+1)st$ column value (i.e. the similarity list) in r is obtained by combining the lists L_1 and L_2 where L_1, L_2 are the lists in the last column of the rows r_1 and r_2 respectively; if $h = h_1 \wedge h_2$ then the lists L_1 and L_2 are combined using the algorithm for the conjunction operator given in the previous subsection; if $h = h_1$ **until** h_2 then the lists are combined using the algorithm for the "until" operator.

Now, we describe the second part. The above inductive approach is used to first compute the similarity table T for g (Recall that $f = \exists x_1 ... \exists x_n\ g(x_1, .., x_n)$). Note that f has no free variables and so we should simply compute a similarity list denoting its similarity value at various video segments. Now consider any video segment u. The actual similarity value of f at u will be the maximum of the similarity values of g at u where the maximum is taken over all possible evaluations ρ for g. Note that each row of T contains a relevant evaluation for g and a similarity list for this evaluation. Now, suppose that u is contained (i.e. contained in an interval of an entry) on the similarity lists of exactly two rows in T. Then, the actual similarity value of f at u is the maximum of the actual similarity value of u in these two entries. In general, the similarity value of f at u is non-zero if u appears in at least one similarity list in T (i.e. is contained in the intervals of an entry of the list), and the actual similarity value of f at u is the maximum of the actual similarity values associated with all such entries.

Let m be the number of rows in T. Now, to actually compute the similarity list of f, we simply take all the lists appearing in different rows and run a modified m-way merge algorithm on these lists. If a video segment appears on multiple lists then we simply take the maximum of all the corresponding similarity values. The algorithm is somewhat more complicated since it has to actually work on the intervals of the associated entries without expanding the interval into a set of ids of video segments, as this might make it very inefficient. The complexity of this part of the algorithm is $O(l \cdot \log m)$ where m is the number of similarity lists and l is their sum.

Now, we analyze the combined complexity of parts 1 and 2 of the algorithm. Assume that $P_1, ..., P_k$ are all the non-temporal predicates appearing in g (if the same predicate appears more than once then we count it as many times). Each of them may have free variables and hence their similarity tables may have more than one row. For each $i = 1, ..., k$, let l_i be the maximum length of a similarity list appearing in any row of the similarity table of P_i, and $\bar{l}$ be the sum of all l_i. The combined complexity of the above algorithm is $O(m\bar{l}(p + \log m))$.

3.3 Algorithms for the full Conjunctive formulas

Now, we briefly describe the algorithm for the full conjunctive formulas. Recall that conjunctive formulas can have two types of variables— object variables and attribute variables. While the object variables are quantified using existential quantifiers as in the case of type (2) formulas, the attribute variables are quantified (or instantiated) using the assignment operator. Now consider a conjunctive formula f given by $\exists x_1 \exists x_2 ... \exists x_n\ g(x_1, ..., x_n)$. Consider a subformula h with free object variables $x_1, x_2, ..., x_k$ and free attribute variables $y_1, ..., y_m$. For each such subformula h of f, we compute a similarity table with $k+m+1$ columns where the first k columns define an evaluation for the object variables, the next m columns give ranges of values for the corresponding attribute variables and the last column is a similarity list.

In the HTL language, predicates containing attribute variables are restricted to be of the form $y_i > q$, $y_i < q$, $y_i \leq q$, $y_i \geq q$ and $y_i = q$ for cases when q is an integer attribute value; for other cases, these predicates are restricted to be of the form $y_i = q$. In all these cases, the values of y_i that satisfy conjunctions of the above type of predicates can be represented as a range of values; here we assume that y_i is of type integer. For this reason, we use range values for attribute variables. In any row, the value of a column for an object variable is an object id, while the value of a column for an attribute variable is a range, and the value of the last column is a similarity list. The value of last column denotes the similarity list for all evaluations in which the object variables take exactly the values given by the corresponding columns, while each attribute variable takes any value belonging to the range specified in the corresponding column.

Now, we inductively define how the similarity tables will be computed. For the case when h is a non-temporal formula we compute the table as before; recall that for different ranges of values for attribute variables we may get different similarity lists. When $h = h_1 \wedge h_2$ or $h = h_1 \,\mathbf{until}\, h_2$, we compute the similarity table as before with the following change; when we use joins on the columns corresponding to the attribute variables their ranges have to be intersecting and the value in the corresponding column in the output is the intersection of the two ranges of the participating rows. Now consider the case when $h = [y \leftarrow q]\ h_1$ where q is an attribute function possibly containing free object variables. Let T_1 be the table corresponding to h_1. The last column of T_1 gives a similarity list and other columns correspond to object and attribute variables (including y) appearing free in h_1. We assume that the value of q is itself given by a table R (called *value table*) whose first few columns, say first k columns, give the values of object variables appearing free in q and whose last but one (i.e. $(k + 1)st$) column gives the value of q and whose last column (i.e. $(k + 2)nd$) is a list of intervals of ids of video segments. Consider a row r whose $(k+1)st$ column has value z and whose last column is the list M; this row denotes that, for an evaluation to the free object variables given by the first k columns, the value of q is equal to z at all the ids contained in the list M.

Now, the similarity table T for the formula h is computed by joining the tables R and T_1 as follows. Consider a row r in R and a row r' in T_1. We join them if their columns denoting the same object variables have equal values and the column corresponding to the value of q in R (i.e. the value in the $(k + 1)st$ column) is contained in the range given by the column for the variable y in r'; in this case, we generate a row in T whose columns for all object variables and attribute variables (other than y) are copied from r' and whose last column (i.e. the similarity list) is the list obtained by merging the list M in r with the similarity list L of the row r'; this merging is carried out as follows. Let I be an entry in the list L and J be an interval on the list M. If the interval of I and J intersect then we generate a entry on the output list whose interval part is this intersection and whose similarity value is same as that from I. The output list contains only such entries, and this list is assigned to the last column of the generated row of T. Note that T will not have any column corresponding to the attribute variable y.

As in the previous subsection, let $\bar{l}$ denote the sum of the lengths of the longest lists appearing in each of the similarity and value tables. The overall complexity of the algorithm is $O(w\bar{l}(p + \log r))$ where w is the number of rows in the similarity table for g and p is the length of f.

The algorithm for extended conjunctive formulas will be given in the full paper.

4. Experimental Results

There is an alternate approach for computing similarity values. This involves translating the formulas into SQL queries. However, such translations become quite complex and the intermediate relations may become quite large.

In this section, we discuss our implementation and some of our experimental results. We have actually implemented two systems. The first system uses the direct algorithms presented in this paper. As part of this system, we have only implemented the algorithms for type (1) formulas. The second system is SQL based, i.e., it uses translations into SQL for computation of the similarity tables for any conjunctive formula. The advantages of taking the second approach are that it permits flexibility, and permits us to take advantage of the query optimization capabilities of the commercial relational database system.

Both these systems have a common front end where the input conjunctive temporal formula f is parsed and its subformulas are identified. Both these systems also take the similarity tables associated with the atomic subformulas of f as input; recall that the atomic subformulas of f are its maximal subformulas that do not have any temporal operators in them. The first system computes the similarity table of a subformula g inductively using the similarity tables for its major constituent subformulas and by using the algorithms presented in the earlier sections. For example, if $g = g_1 \wedge g_2$ or $g = g_1 \,\mathbf{Until}\, g_2$ then it computes the similarity table for g using the tables for g_1 and g_2 by employing the algorithms presented earlier. The second system, i.e. the SQL based system, first generates a sequence of SQL queries which take as inputs the tables for g_1 and g_2 and output the table corresponding to g, and then executes the sequence of SQL queries. The automatic generation of SQL queries is non-trivial and the interested is referred to [22].

4.1 Test Cases

The first video that we considered was titled "The making of the Casablanca". The movie was about 30 minutes in duration. Firstly, the movie was segmented into smaller sequences(called shots) using a method called *cut-detection* [21, 11]. After cut-detection, we had 50 shots. Then the meta-data(which provides information regarding the contents of the shot and is used to answer user queries) for each shot was entered into a pictorial database system.

In the following example all the queries correspond to *type(1)* formulas. Further, the following atomic predicates(corresponding to the pictures in the query) are used in the query. The similarity tables for these atomic predicates are computed using an existing picture retrieval system [2, 27]. The data corresponding to the different shots is fed into the picture retrieval system considering each shot as a single picture. The atomic predicates are posed as queries to the picture retrieval system and the similarity tables computed by the picture retrieval system are fed as input to our video retrieval system. In all the tables, the last column gives the actual similarity value; the maximum similarity value is omitted since all the rows in a table have identical values in this column.

Moving_train: This predicate states that there is a moving train in the shot. The similarity table for this predicate is shown in table *1*.

Man_Woman: This predicate asserts the presence of a man and a woman. The similarity table for this predicate, generated by the picture retrieval system, is given table in *2*. The entries in this table having lower similarity values correspond to pictures/shots containing two men instead of a man and a woman.

Start_id	*End_id*	*Similarity_value*
9	9	9.787

Table 1. Moving_Train

Start_id	*End_id*	*Similarity_value*
1	4	2.595
6	6	1.26
8	8	1.26
10	44	1.26
47	49	6.26

Table 2. Man_Woman

We first considered "Query 1" given by the following formula.

{ Man_Woman $\wedge$ { **eventually** Moving_train }}

We used both approaches, i.e. the direct method and the SQL-based method to compute the similarity values for this query. Both approaches produced identical final values as well as identical intermediate similarity tables. The intermediate result corresponding to { **eventually** Moving_train } is given in table *3*. The final result corresponding to the main query is given in table *4*.

Start	*End*	*Sim*
1	9	9.787

Table 3. Result of eventually operation in Query 1

4.2 Performance Comparison of the two Approaches

In the previous subsection, we have illustrated the result of a query through a simple example whose size is small. But a real world video application needs to handle large amounts of data. In this subsection, we compare the response times of the two approaches on larger data sizes than the "making of Casablanca". Since, we do not have access to large amount of real world data, we compared the performance of the two approaches on randomly generated data.

The performance of the two approaches, to a large extent, depends on how they handle the basic connectives $\wedge$ and **until**. For this reason, we compare their performance on the two temporal formulas $P1 \wedge P2$, and $P1$ $\mathbf{until}$ $P2$. In both these cases, the inputs to the algorithms are the similarity tables for $P1$ and $P2$. We ran both systems for different input sizes. The time given for the direct approach includes the time required to read the similarity tables of $P1$ and $P2$ from the secondary storage, the time required to sort the tables on the start ids and the running time of the algorithm. For the SQL-based method, the time required is the time for executing the sequence of SQL queries generated on the similarity tables of $P1$ and $P2$. The SQL-based system used Sybase relational database.

Start	*End*	*Sim*
1	4	12.382000
6	6	11.047000
8	8	11.047000
5	5	9.787000
7	7	9.787000
9	9	9.787000
47	49	6.260000
10	44	1.260000

Table 4. Final result of Query 1

The performance results are given in the tables 5 and 6. In these tables, the first column corresponds to the size, which is the number of shots in the movie; approximately about one tenth of these shots satisfy the atomic predicates $P1$ and $P2$. The second and the third columns give the time taken (in seconds) by the direct approach and the SQL-based approach respectively. In the direct approach, we tried different sorting algorithms. The numbers given in the table are for Merge sort.

From these tables, it is clear that the performance of the direct method is much better than the SQL-based approach. Inefficiency of the SQL-based approach is partly due to the fact each of the formulas is translated into many SQL queries. It is also to be observed that the time taken by the direct method increases linearly with the size which is in confirmity with our complexity analysis. The growth rate of the SQL-based approach is less than linear; this may be the effect of the overhead. In addition to the two basic formulas, we also analyzed the performance of the two approaches on two other more complex formulas. The results for these more complex cases are consistent with those for the simpler formulas and are left out due to lack of space.

Size	*Direct Approach*	SQL-based
10000	2.36	61.23
50000	12.37	131.76
100000	24.43	179.45

Table 5. Perf Results for $P1 \wedge P2$

Size	*Direct Approach*	SQL-based
10000	1.46	42.14
50000	7.35	99.72
100000	14.97	134.63

Table 6. Results for "P1 until P2"

5. Conclusion

In this paper we have introduced a query language for specifying queries which is based on the hierarchical and temporal nature of the video data. We have also given efficient similarity based retrieval algorithms that work on top of an existing picture retrieval system. We have also given an SQL-based approach.

Some preliminary experimental results, showing the effectiveness of our methods, have been given. Our retrieval methods are for a subclass of the language HTL (called extended conjunctive formulas) that we introduced in this paper. We believe that most queries of practical interest can be expressed in this subclass. We have done a comparison of the two approaches on randomly generated data. Our experimental results show that the direct approach performs much better than the SQL-based approach using a Sybase relational database system on SUN workstations on the sizes of data considered by us.

As part of future research, we would like to investigate the extension of the above methods to the full language. It will also be worthwhile to investigate other similarity functions, other than the *fractional similarity function* that we have discussed in this paper. We also feel that a user-friendly graphical interface for specifying the temporal queries will be very useful. Of course, one of the most important problem is the automatic generation of the meta-data. Much work is being done by the image-processing and vision community in this regard, and is beyond the scope of this paper.

Acknowledgements: We are thankful for Dr. Francis Quek and Robert Bryll for providing us some real experimental data.

References

[1] S. Al-Hawamdeh and et al. Nearest neighbor searching in picture archival system. *International Conference on Multimedia Information System*, 1991.

[2] A. Aslandogan, C. Thier, C. T. Yu, and et al. Implementation and evaluation of score(a system for content based retrieval of pictures). *IEEE Data Engineering Conference*, March 1995.

[3] C. Breiteneder, S. Gibbs, and D. Tsichritzis. Modeling'of audio/ video data. *Proc. ER conference*, 1992.

[4] T. Chua and L. Ruan. A video retrieval and sequencing system. *ACM Transactions on Information Systems*, October 1995.

[5] Y. Day and et al. Object-oriented conceptual modeling of video data. *IEEE Data Engineering*, 1995.

[6] N. Dimitrova and F. Golshani. Rx for semantic video database retrieval. *ACM Multimedia conference*, 1994.

[7] M. Flickner, H. S. Sawhney, W. Niblack, and et al. Query by image video content: The qbic system. *Computer*, Sept. 1995.

[8] C. Faloutsos and et al. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(1), 1994.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time series databases. *ACM SIGMOD*, 1994.

[10] A. Gupta. *Visual Information Retrieval Technologies, A Virage perspective*. White Paper, Virage Incorporated, 1995.

[11] A. Hampapur, R. J. R, and T. Weymouth. Production model based digital video segmentation. *Multimedia Tools and Applications*, 1, 1995.

[12] E. Hwang and V. S. Subrahmanian. Querying video libraries. *Technical Report, Department of Computer Science, University of Maryland*, 1995.

[13] R. Jain, K. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw Hill, 1995.

[14] R. Hjelsvold and R. Midtstraum. Modelling and querying video data. *VLDB*, 94.

[15] T. Little and Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE TKDE*, 1993.

[16] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems–Specification*. Springer-Verlag, 1992.

[17] W. Niblack, R. Barber, and W. Equitz. The qbic project: Querying images by content using color, texture, and shape, spie 1980. *Storage And Retrieval for Video and Image Databases*, Feb. 1993.

[18] A. Nagasaka and Y. Tanaka. Automatic video indexing and full video search for object appearances. *Second working Conference on Visual Database Systems*, Oct. 1991.

[19] E. Oomoto and K. Tanaka. Ovid: Design and implementation of a video-object database system. *IEEE TKDE*, 1993.

[20] O. Kiyotaka and T. Yoshinobu. Projection-detection filter for video cut detection. *Multimedia Systems*, 1, 1995.

[21] F. Quek. Vector coherence maps. *Technical Report, Electrical Engineering and Computer Science Department, University of Illinois at Chicago*, 1995.

[22] R. Venkatasubramanian. Similarity based retrieval of videos. *M.S. thesis, Department of Electrical Engineering and Computer SCience, University of Illinois at Chicago*, June 1987.

[23] I. K. Sethi and R. Jain. Finding trajectories of feature points in a monocular image sequence. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9, 1987.

[24] A. P. Sistla and O. Wolfson. Temporal triggers in active database systems. *IEEE TKDE*, July 1995.

[25] A. P. Sistla and C. Yu. *Multimedia Database Systems*. Springer, 1996.

[26] A. P. Sistla, C. Yu, and R. Haddad. Reasoning about spatial relationships in picture retrieval systems. *VLDB*, 94.

[27] A. P. Sistla and C. Yu. Similarity based retrieval of pictures using indices on spatial relationships. *VLDB*, 95.

[28] Y. Tonomura, A. Akutsu, and et al. Structured video computing. *IEEE Multimedia*, 1, 1994.

[29] J. Y. A. Wang and E. H. Adelson. Spatio-temporal segmentation of video data. *Proceedings of SPIE: Image and Video Processing II*, 2182, Feb. 1994.

[30] R. Weiss, A. Duda, and D. K. Gifford. Composition and search with video algebra. *IEEE Multimedia*, 1995.