Temporal Query Processing for Scene Retrieval in Motion Image Databases

Junichi Takahashi

IBM Research, Tokyo Research Laboratory
5 19, Sanbancho, Chiyoda ku, Tokyo 102, JAPAN

Abstract

This paper describes a temporal query processing algorithm for efficient scene retrieval in motion image databases. A motion image logically consists of sequential frames, and a scene, defined as an arbitrary portion of the frame sequence, is identified by a pair of numbers representing the initial and final frames of the scene. When the frame numbers are mapped onto a time-axis, a scene object can be represented as a record that has two time attributes for identifying the time interval of the scene object, as well as other attributes for describing scene properties. The proposed query processing method is intended to allow efficient processing of n-way temporal join in searching for overlaps and extensions of multiple intervals by comparison of time attributes for the join. Experiments show that the processing cost of our method increases linearly with respect to the number of compared records, whereas that of the conventional two-way nested-loop method increases exponentially.

1 Introduction

Motion pictures have penetrated widely and deeply into our societies owing to the recent popularization of audio and video equipment and the rapid development of multi-media and interactive technologies [8, 9]. The huge volume of collections of motion pictures makes it impossible to access motion pictures simply by depending on human memory, or on visual scanning of entire collections. The importance of motion image databases for retrieving target scenes of interest rapidly and with minimum effort is definitely recognized in museums, media organizations such as broadcasting centers, and private homes [12]. Because of the temporal properties of continuous media such as motion pictures, motion image databases are an important application of temporal databases, and there has been much recent discussion of technical issues related to motion image databases [13].

A motion image logically consists of sequential frames, each of which is a two-dimensional still image. A scene, defined as an arbitrary portion of a frame sequence, is identified by a pair of numbers representing the initial and final frames of the scene. When the frame number of a motion image is mapped onto a time-axis, a scene object can be represented as a record that has two time attributes as well as other properties for describing the contents of scenes, such as characters and objects appearing in the scenes, or events and background details. When a user wants to retrieve scenes of interest quickly and randomly by specifying the conditions of these scene properties, searching for overlaps and extensions of multiple intervals in an image database are important operations.

So far, research on temporal databases has mostly addressed temporal representations of the time properties of real-world objects that have time-varying properties, as well as issues related to versions [15], operations on the objects [4, 5, 21, 23], temporal query languages [1, 10, 17, 19, 22, 24], and physical file organization [2, 16]. Access methods for temporal databases [6], and temporal query processing [14, 20] have been less frequently addressed, especially for n-way temporal join [11].

Temporal operations on time intervals must include evaluation of temporal relationships among the intervals [3], such as before, after, overlapping, and included. The crucial problem in temporal join processing in motion image databases is that a join is likely to be processed by using a two-way nested-loop algorithm, because both the initial and final frame numbers of different scenes should be compared simultaneously to evaluate the temporal relationships of two intervals. This is why efficient query processing is required for temporal operations such as comparison of temporal attributes.

The query processing method proposed in this paper is intended to allow efficient processing of n-way temporal join. The algorithm is described in the context of the relational representation of temporal objects, but it can also be applied to other temporal models, such as temporal ER databases [7] and temporal object-oriented databases [18].
2 Temporal representation of motion image databases

We can allocate a sequential frame number to each frame of a motion image. As the example in Figure 1 shows, a scene, \(A\) for instance, is an arbitrary portion of a motion image, and consists of a sequence of frames. A scene can be expressed as a pair consisting of the initial frame number and the final frame number. Different scenes may share frames, as scenes \(A\) and \(C\) do in this example. Processing for scene retrieval requires a pair of initial and final frame numbers to be found.

![Figure 1. Scene objects in a motion image](image)

Scene retrieval methods can be divided into two categories according to whether or not they use structural models. Higher efficiency in the management and retrieval of motion image databases is expected to result from the use of structural models [12], as the example in Figure 2 shows. A motion picture can be logically segmented into shorter scenes whose shortest scenes are called cuts, according to criteria such as physical correlations and/or changes in semantics. Each scene has one or more still frames, called representative frames or RFs, which typically represent the entire frame sequence.

The structural models can be grouped into two types with respect to time sequential and hierarchical models. The latter type describes the hierarchical structures of scenes in terms of the story, or scenario, of the motion image. In a scene hierarchy, a scene is represented as a set of nodes each of which stores the properties of the scene, and a set of links each of which represents a parent-child relationship between two scenes. On the other hand, the time sequential model [12] provide users with a flexible and easy-to-understand query specification of scene retrieval, in the sense that a scene structure can be regarded in terms of temporal relationships of pairs of intervals on a time-axis, such as before, after, overlapping, and extension.

Omoto et al. model a motion image as a set of semantic scene objects in an object-oriented database, with a pair of numbers representing the initial and final frames of each scene, as well as other scene properties allocated by a user [18].

![Figure 2. Example of a structural model of a motion image](image)

Though their model of a motion image is very natural and useful as a mean for retrieving scenes of interest by matching scene properties, it can only be used to search for scenes that have been selected from a collection of pre-defined scene objects. In their model, it is inconvenient to add or modify scene properties to or from an arbitrary portion of a motion image, because all of the scene objects must be redefined each time update operations are performed. For example, if there is a scene \(S_1 = [100,300]\) with the property \(K_1\), and the user wants to add the property \(K_2\) to scene \(S_2 = [200,400]\), he has to redefine the following scenes: \(S_1' = [100,200]\) with \(K_1\) and \(K_2\), and \(S_2' = [300,400]\) with \(K_2\).

In this paper, we assume a time-sequential representation of scene properties for flexible indexing of arbitrary portions of motion images [12]. Figure 3 is a sample of a time chart showing the periods of allocation of properties to scenes. A Boolean query can be specified by using an arbitrary number of properties with any combinations of the logical operators AND, OR, and NOT. For example, target scenes that have both properties \(K_1\) and \(K_2\) can be retrieved by searching for the overlapping intervals of \(K_1\) and \(K_2\) in this time chart.

For the time-sequential representation of scene properties, we store a set of an initial frame number \(F_i\) and a final frame number \(F_f\) and a property \(K\) in a record \(R = [F_i, F_f, K]\) \(F_i < F_f, 0 < i \leq m\). A temporal table \(D(\text{start}, \text{end}, \text{property})\) is a collection of these records, each of which has two temporal attributes for the initial and final frame numbers. Figure 4 shows a sample of table \(D\).
In this representation of scene properties, operations for adding and updating properties of scenes can be performed flexibly, because each scene property is described in a unique record, and modification of a property of a particular scene does not affect the allocation of other properties to the scene, or that of any properties of other scenes.

3 Problems in temporal query processing

Snodgrass proposed a method of searching for overlaps and extensions of intervals, based on temporal join, in the context of temporal databases [22]. The join operator is translated into some conventional join conditions that compare both the initial and final numbers of two intervals at a time. For the table shown in Figure 4, a query retrieving overlaps between intervals with property $K_1$ and those with property $K_2$ can be processed by translating the query into the following self-join SQL statement:

```sql
select d2.start, d1.end
from D d1, D d2
where d1.property in ($K_1$, $K_2$)
and d2.property in ($K_1$, $K_2$)
and d1.property=d2.property
and d1.start<d2.start
and d1.end<d2.start
and d1.end<d2.end
union
select d2.start, d2.end
from D d1, D d2
where d1.property in ($K_1$, $K_2$)
and d2.property in ($K_1$, $K_2$)
and d1.property=d2.property
and ((d1.start<d2.start and d1.end>d2.end)
or (d1.start=d2.start and d1.end>d2.end)
or (d1.start=d2.start and d1.end=d2.end and d1.property>d2.property))
```

This query includes disjunctive temporal conditions, because there are three types of overlapping between two intervals: (a) intersect, (b) contains or contained, and (c) same, as the example in Figure 5 shows. The first select clause in the above statement includes a condition on the time attributes start and end for intersecting intervals:

```sql
d1.start<d2.start and
d1.end<d2.start and
d1.end<d2.end
```

In the same way, the second select clause includes three disjunctive conditions for containing, contained, and same intervals, respectively. In this case, a merge-sort or hash-join method cannot be applied effectively for processing the temporal join conditions. Because the two time attributes start and end are compared simultaneously, these conditions should be evaluated on the basis of a nested-loop algorithm that requires $n^2$ record comparisons, where $n$ is the number of records in the table to be joined. The problem is that the search cost increases exponentially relative to the number of compared records.

Elmasri proposed a time index access method that can be used to improve the performance of temporal selection, temporal projection, aggregate functions, and temporal join [6]. Because query processing with this index access method is two-way for temporal join operations, it requires iteration of the two-way processing to implement the n-way join for processing Boolean queries in a scene retrieval described such as that in the previous section.

Gunadhi et al. proposed a temporal query processing algorithm that is more efficient than the nested-loop based algorithm [11]. In their method,
two sets of records are separately sorted in ascending order of the starting numbers of intervals. These sorted records are sequentially accessed in order to match temporal attributes without duplicate access to the same records. The drawback of this algorithm is that it is intended for two-way join, and only for queries in which exactly two properties are specified for the search condition.

4 Temporal query processing

4.1 Outline of query processing

An outline of the query processing procedure is given here, using the example of the table shown in Figure 4. Assume that we want to retrieve intervals that have all the properties $K_1$, $K_3$, and $K_4$. Query condition $Q$ is expressed in terms of a Boolean expression: $Q = K_1$ AND $K_3$ AND $K_4$.

First, the records in table $D$ are restricted to those that have been allocated one of the properties $K_1$, $K_3$, or $K_4$. An intermediate table $D'$, shown in Figure 6, is created, which gives a set $A$ including six intervals:

$$A = \{[20,130], [40,80], [50,100], [150,180], [160,230], [170,220]\}.$$  

If the whole interval $Z$ is $[0,300]$ in this example, we obtain candidate intervals $I(i)$ for evaluating the query condition $Q$:

$$I(i) = [L(i), L(i + 1)] \quad (1 \leq i \leq |A| \times 2 + 1),$$

where $L(i)$ is the $i$th element of the list composed of the following $|A| \times 2 + 2$ elements of the initial or final number of intervals in set $A \cup Z$:

$$0, 20, 40, 50, 80, 100, 130, 150, 160, 170, 180, 220, 230, 300.$$  

Note that the starting number $L(i)$ of the $i$th interval $I(i)$ is incremented by 1 if $L(i)$ is derived from the final number of any interval in list $A$, and that the final number $L(i + 1)$ is decremented by 1 if $L(i + 1)$ is derived from the starting number of any interval in list $A$. In this case, we get the following 13 candidate intervals:

$$[0,19], [20,39], [40,49], [50,80], [81,100], [101,130], [131,149], [150,159], [160,169], [170,180], [181,220], [221,230], [231,300].$$

Now, a truth table, shown in Figure 7, is created to evaluate whether each property specified in the query is allocated to each of these candidate intervals. The resultant intervals, $[50,80]$ and $[170,180]$ in this example, are given if the truth value of $Q$ is TRUE for any of the candidate intervals in this truth table.

$$\begin{array}{|c|c|c|c|c|}
\hline
\text{Interval} & [0,19] & [20,39] & [40,49] & [50,80] \\
\hline
K_1 & F & T & T & T \\
K_3 & F & F & F & T \\
K_4 & F & F & T & T \\
Q & F & F & F & T \\
\hline
\end{array}$$

$$\begin{array}{|c|c|c|c|c|}
\hline
\hline
0 & T & T & F & F & T \\
1 & T & F & F & T & T \\
F & F & F & F & F & F \\
F & F & F & F & F & F \\
\hline
\end{array}$$

$$\begin{array}{|c|c|c|c|}
\hline
\hline
T & T & T & F & F \\
T & T & F & F & F \\
T & T & F & F & F \\
T & T & F & F & F \\
\hline
\end{array}$$

Figure 7. Truth table for candidate intervals

4.2 Query processing procedure

This procedure assumes that the temporal table $D$ has two temporal attributes for the initial and final number of intervals, and an attribute for the property of the intervals. The input of this procedure is a Boolean query $Q$ composed of $m$ properties of target intervals. The output of the procedure consists of intervals that satisfy the query condition. In the query results, if the final number of an interval is adjacent to the final number of another interval, those two intervals are merged into the shortest interval that includes both the intervals. The following is a description of the query processing procedure:

Step 1. Create a logical formula $P(P_1, P_2, ..., P_m)$ by replacing property $K_i$ in query $Q$ with logical variable $P_i$.

Step 2. Search records in table $D$ that have one of the properties specified in query $Q$, and store the records in an intermediate table $D'$ after sorting them according to the initial number $F_i$.

Step 3. Prepare the following work areas in main memory, and initialize them:

- Target interval buffer: $S_r := [F_i, F_i]$
b) Candidate interval buffer: \( S_c := [F_{ia}, F_{fc}] \)
where the initial values are
\( F_{ia} = F_{fc} = (F_i \text{ of } K_i) - 1 \)
c) Current interval buffer: \( S_i := [F_{mi}, F_{fi}] \)
\((0 < i \leq m)\)
d) Input buffer: \( B := [F_{ia}, F_{bi}, K_i] \)
e) Record buffer: \( B' := [F_{fr'}, F_{br'}, K_{br'}] \)

Step 4. If record buffer \( B' \) is empty, read a record into input buffer \( B \) from intermediate table \( D' \). If there is no record to read, raise an EOF flag. If record buffer \( B' \) is not empty, copy \( B' \) to input buffer \( B \), and clear \( B' \).

Step 5. Find the number \( i \) such that \( K_i = K' \), and compare the final number of the intervals of the candidate interval and the current interval for \( i \).

\[
\text{if } F_{fc} \geq F_{fi} \\
\text{then begin } F_{wi} := F_{wi}; F_{fwi} := F_{fi}; \text{ end} \\
\text{else } B' := B; 
\]

Step 6. Update the candidate interval buffer \( S_c \) as follows. Set the next number of the current final number of \( S_c \) to the new initial number of \( S_c \). Set the minimum initial or final number among the current intervals that is larger than the final frame number to the new final number.

\[
F_{ic} := F_{ic} + 1 \\
F_{fc} := \min \{F_{iw} - 1, F_{frk}\} \\
\text{where } F_{iw} > F_{fi}, F_{frk} > F_{fc} 
\]

Step 7. Set the logical variables \( P_i \) by checking inclusion between the candidate interval and the current interval for \( i \).

\[
\text{if } F_{ic} \geq F_{fi} \text{ AND } F_{fc} \leq F_{wi} \\
\text{then } P_i := \text{TRUE} \\
\text{else } P_i := \text{FALSE}; 
\]

Step 8. Evaluate the logical formula \( P \) by using values of \( P_i \). If the logical formula \( P \) is \( \text{TRUE} \), write the target interval \( S_t \) after checking the connection between the candidate interval and the target interval, which is found in the previous loop.

\[
\text{if } F_{iw} = F_{fi} + 1 \\
\text{then } F_{ft} := F_{fc}; \\
\text{else } S_t := S_c; 
\]

Step 9. If the EOF flag is not raised, go to step 4.

In the above procedure, steps 4 to 9 are iterated for each candidate interval. Because the intermediate table is sorted in ascending order of the initial number of intervals in step 2, it is possible to evaluate the truth value for the candidate intervals by making one sequential search of the intermediate table.

### 5 Experimental results

The search cost in the proposed temporal query processing method with respect to disk pages accessed includes: (1) the number of pages occupied by temporal table \( D \), (2) the number of pages occupied by intermediate table \( D' \), and (3) the sorting cost for the intermediate table. Figure 8 shows a comparison of the response time in our method and that in the conventional two-way method, which utilizes nested-loop join processing. The search cost in our method is proportional to the number of records in the intermediate table \( D' \), because the target intervals are found by accessing the intermediate table only once. If the intermediate table is small enough to be accommodated in main memory rather than secondary storage, costs (2) and (3) can be ignored. It is possible to improve cost (1) by using a \( B^+ \)-tree index on the property attribute of the temporal table \( D \) in restricting the records on the attribute.

In the conventional method, the records in the temporal table can be restricted to same records that appear in the intermediate table in our method with the properties specified in a query. But the search cost still increases exponentially rather than linearly with respect to the number of those records, because they are accessed repeatedly in each loop of the nested processing of time attribute comparison. Moreover, a query condition consisting of three or more properties must be processed by iterating the algorithm for two-way join sub-query as many times as the number of properties in the query, so the search time also

<table>
<thead>
<tr>
<th>Table size in records</th>
<th>Proposed method</th>
<th>Two-way method</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intermediate table size)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query: ( Q_1 = K_1 \text{ AND } K_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>472 (218)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>944 (436)</td>
<td>14</td>
<td>29</td>
</tr>
<tr>
<td>1888 (872)</td>
<td>27</td>
<td>99</td>
</tr>
<tr>
<td>Query: ( Q_2 = K_1 \text{ OR } K_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>472 (218)</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>944 (436)</td>
<td>14</td>
<td>37</td>
</tr>
<tr>
<td>1888 (872)</td>
<td>27</td>
<td>137</td>
</tr>
<tr>
<td>Query: ( Q_3 = K_1 \text{ AND } K_2 \text{ AND } K_3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>472 (218)</td>
<td>7</td>
<td>N/A</td>
</tr>
<tr>
<td>944 (436)</td>
<td>14</td>
<td>N/A</td>
</tr>
<tr>
<td>1888 (872)</td>
<td>27</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Figure 8. Search cost comparison**
increases in proportion to the number of properties in the original query.

6 Conclusion

We proposed a temporal query processing algorithm for n-way temporal join in temporal databases, which can be applied to motion image databases for flexible and random search of scenes of interest. The algorithm is superior to the conventional two-way nested-loop method in terms of memory cost, flexibility, and performance. We also confirmed the performance of our query processing method when we built a prototype of a motion image database system [12]. We plan to make feasibility studies of the motion image database system for practical applications in a museum, in which our temporal query processing is used for flexible and efficient scene retrieval.

References