Journal of Visual Languages and Computing (1998) 9, 375–397
Article No. vl980089

Video Data Indexing by 2D C-Trees

FANG-JUNG HSU,*† SUH-YIN LEE* AND BAO-SHUH LIN‡

** Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, HsinChu, Taiwan 30050, Republic of China, email: fjhsu@info4.csie.nctu.edu.tw.*
‡Computer & Communication Research Laboratories, Industrial Technology Research Institute, HsinChu, Taiwan

Accepted 5 May 1998

Video data contains a large amount of spatial and temporal information. The changes of video frames are quite useful for motion analysis and cannot be provided by other media easily. Chang *et al.* had proposed an effective 2D string approach for spatial indexing of image data. In this paper, we extend this iconic approach and apply it to video data indexing. We propose 2D C-trees to represent the spatial content within individual frames. A video sequence can then be represented and indexed by a temporal set or an ordering set of 2D C-trees. The similarity retrieval of video matching problem becomes the problem of video sequence matching by computing the similarity, or the minimum cost of matched frames. We present three schemes, full-sequence matching, segment matching and subsequence matching, for video information retrieval. The matching schemes can be modified and extended to approximate sequence matching by computing the partial distance for providing a comprehensive retrieval of video data. A prototype video information system is also developed to validate the effectiveness of video data indexing by 2D C-trees.

Notation

A_b	the begin-bound of object A
A_e	the end-bound of object A
R	the root of a 2D C-tree
s_i	the ith immediate descendant of node S
ε	empty-node
$\mathfrak{J}$	a rooted tree
$e_1 \ldots e_k$	the editing operations
Δ	the cost function of editing operation
$\delta(\mathfrak{J}_1, \mathfrak{J}_2)$	the tree distance between tree $\mathfrak{J}_1$ and tree $\mathfrak{J}_2$
$\eta(\mathfrak{J})$	the number of symbols in tree $\mathfrak{J}$
$\|w\|$	the size of a frame w
$\delta(r, q)$	the frame distance between reference frame r and query frame q
V	the reference sequence

† Corresponding author.

1045-926X/98/040375+23 $30.00/0

v_i	the ith frame of reference sequence V
U	the query sequence
u_j	the jth frame of query sequence U
$D[i, j]$	the minimum cost of transforming from $v_1 v_2 \ldots v_i$ to $u_1 u_2 \ldots u_j$
$\Psi(V, U)$	the full-sequence matching distance between sequences V and U
τ	a matching threshold
$d[i, j]$	the partial cost of transforming from $v_1 v_2 \ldots v_i$ to $u_1 u_2 \ldots u_j$
$\psi(V, U)$	the subsequence matching distance between sequences V and U
$\gamma(r, q)$	the partial distance between reference frame r and query frame q

1. Introduction

WHEN MANAGING a large amount of data of any kind, indexing is useful for rapid retrieval. As we move towards on-line digital libraries and global access to digital data, indexing is becoming more important in database management systems [1]. Video is a medium with high complexity that contains a large amount of spatial and temporal information. It can provide more information than text, graphic and even image. The information related to the position, distance, temporal and spatial relationship is included in the video data implicitly. In current video database systems, fundamental techniques such as keyword-based searching [2] and hierarchical video icon browsing [3], are provided for user query. Most of the previous researches in video data are focused on motion and scene analysis [4]. Very little work has been done on the design of index structures that characterize spatial and temporal attributes for video databases.

Video frame sequences, as compared with a single video frame, contain information about the dynamic aspects of a recorded scene. Especially, the changes of video objects in equally divided temporal intervals are quite useful for motion analysis and cannot be provided easily by other media. For example, we can ask a query like: *retrieve all the video sequences in which there is a running athlete holding a basketball, crossing over the field, jumping up, hanging over the basket and then slamming.* Much spatial and temporal information of video sequence needs to be extracted to serve this query. These spatial and temporal relations play the indexing role in content-based video information retrieval [5–7]. Unfortunately, the spatio-temporal information of video data cannot be extracted automatically without human interaction with the current techniques in image/video understanding and recognition. Even if the spatial and temporal relationships among objects or semantic units can be inferred from an objects location and duration, state-of-the-art software for manipulating video does not 'know' about such objects [8], for example, what a basketball is. Nevertheless, an iconic approach with human involvement can compensate the difficulty and deficiency.

The iconic approach by 2D strings for spatial indexing was initially proposed by Chang *et al.* [9] to represent symbolic pictures. First, the objects in original pictures are identified after preprocessing by either image processing and pattern recognition techniques or human involvement. Then the orthogonal relation objects with respect to other objects are generated and iconized. After all the iconic objects have been processed, the symbolic picture that preserves the spatial relationships among objects in the original picture is encoded as a 2D string. The problem of pictorial information

retrieval then becomes the problem of 2D subsequence matching [10]. This approach thus allows a natural way to construct iconic spatial indexing for pictures.

To represent the spatio-temporal information, the approach of iconic image indexing is extended to video data indexing. We investigate a new knowledge model to represent the spatial content within individual frames and construct a temporal set for a video sequence. We attempt to solve this video information retrieval problem in three areas [3]:

(1) Developing a video indexing structure that characterizes the spatio-temporal information of video content.
(2) Applying knowledge representation techniques to the development of index construction and retrieval tools.
(3) Developing a video information retrieval environment for interacting with video objects.

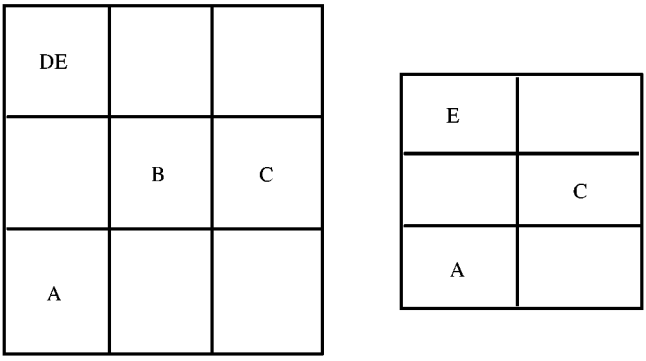
In the paper, the spatial indexing for pictures by 2D strings are briefly introduced in Section 2. Then we propose a new knowledge representation, 2D C-tree [11], to be the spatial indexing for pictures in Section 3. The 2D C-tree is employed for single frame matching on similarity retrieval. Section 4 describes how 2D C-trees are extended to video sequences matching in video information retrieval. We propose three matching schemes: full-sequence matching, segment matching and subsequent matching. The algorithms of the matching schemes are developed in Section 5. The video sequence matching algorithms are modified and extended to approximate sequence matching for providing a comprehensive video information retrieval methodology discussed in Section 6. In Section 7, we also present a prototype system in our video information retrieval project. Finally, conclusions and future works are summarized.

2. Spatial Indexing

The 2D string approach for spatial indexing was initially proposed by Chang *et al.* [9] to represent symbolic images. Three spatial relation operators '$<$', '$=$', and '$:$' are employed in 2D strings. The operator '$<$' denotes the 'left–right' or 'below–above' spatial relation. The operator '$=$' denotes the 'at the same spatial location as' relation. The operator '$:$' denotes the 'in the same set as' relation. The symbolic picture f_1 in Figure 1(a) can be represented as the 2D string $(A = D : E < B < C,\ A < B = C < D : E)$ or as $(A = DE < B < C,\ A < B = C < DE)$ where the symbol '$:$' can be omitted and is omitted.

The 2D string representation is also suitable for formulating picture queries. In fact, we can imagine that the query can be specified graphically, by drawing an iconic image on the screen of a computer. The graphic representation, called *icon sketch*, can be translated into the 2D string representation. For example, we may want to retrieve images satisfying a certain icon sketch q_1 as in Figure 1(b). Then q_1 can be translated into the 2D string $(A = E < C,\ A < C < E)$. This query string is a sub-string of the 2D string representation of the example image f_1. The problem of image retrieval then becomes the problem of 2D string subsequence matching.

However, the spatial operators of 2D strings are not sufficient to give a complete description of spatial knowledge for images of arbitrary complexity. Many extended



(a) f_1 (b) q_1

Figure 1. A symbolic image and a query sketch

Table 1. The definition of characteristic spatial operators

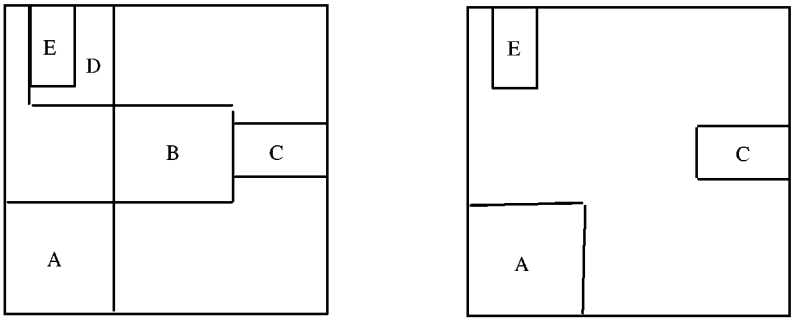
Notation	Condition	Meaning
$A<B$	$A_e<B_b$	A disjoins B
$A\|B$	$A_e=B_b$	A is edge to edge with B
$A=B$	$A_b=B_b$, and $A_e=B_e$	A is the same as B
$A[B$	$A_b=B_b$, and $A_e>B_e$	A contains B and they have the same begin-bound
$A]B$	$A_b<B_b$, and $A_e=B_e$	A contains B and they have the same end-bound
$A\%B$	$A_b<B_b$, and $A_e=B_e$	A contains B and they do not have the same bound
A/B	$A_b<B_b<A_e<B_e$	A is partly overlapping with B

Note: The notations A_b and A_e (B_b and B_e) denote the values of begin- and end-bound of object $A(B)$, respectively.

representations were proposed to handle more types of relations between pictorial objects, but they are not economic for complex images in terms of storage space efficiency and navigation complexity. Lee and Hsu [12] proposed 2D C-string representation with a set of spatial operators and a more efficient cutting mechanism. They employed a characteristic set of spatial operators illustrated in Table 1 to give a complete description for images of arbitrary complexity.

Basically, the 2D C-string approach performs a cut to handle the cases of objects with partly overlapping. The *global* operators '$<$' and '$|$', which are employed in the original 2D string approach, handle the cases of non-overlapping. The extended operators '$=$', '[', '%', and ']', called the *local* operators, and a pair of separators '( )' handle the cases of overlapping. The picture f_2 in Figure 2(a) is similar to f_1 in Figure 1(a) except that the objects in f_2 are non-zero-sized objects as opposed to point objects in f_1. The 2D C-string representation of the picture f_2 is $(A]D[E|B|C, A|B\%C|D]E)$. It is noted that all the objects in f_2 keep intact without cutting because the case of partly overlapping does not happen.

The 2D C-string is efficient in representing and manipulating images, but it is not suitable for solving the problem of 2D string subsequence matching. For example, we



(a) f_2 (b) q_2

Figure 2. A symbolic image with non-zero sized objects and a query sketch

use a query sketch q_2 as in Figure 2(b). The 2D C-string representation of the query image is ($A\,\%E < C$, $A < C < E$). Then this query string of q_2 is quite different from the example string of f_2 due to the spatial operators. The former string is not the substring of the latter string any longer. Unfortunately, these operators are needed to handle the global and local relations among symbolic objects in a 2D C-string and cannot be omitted.

Though the inference of the spatial relations between objects from a given 2D C-string in spatial reasoning can be solved by using the ranking mechanism [13], the computation of object ranks in a 2D C-string is somewhat complicated. Moreover, all the spatial relationships of objects pairs, which is $O(n^2)$ for n objects in an image, are required to be reasoned first by adopting the 2D longest common subsequence algorithm. The algorithm for similarity retrieval actually finds a maximum clique [13] and becomes an NP-complete problem, though there are some polynomial-time algorithms for the average case. Therefore, we explore a more effective representation for spatial indexing of images.

3. 2D C-tree

We first introduce the basic structure of a 2D C-tree [11], which is an ordered labeled tree. The 2D C-tree representation still employs the sparse cutting mechanism to handle the case of symbolic images with partly overlapping objects. The cutting mechanism performs the required cuts to get rid of the ambiguity incurred due to partly overlapping. After cutting, an image is partitioned to some portions between two neighboring cuts. All the portions are sequentially linked to a root, R, which is initialized to represent the margin or boundary of the area covered by a given image.

The 2D C-tree representation was developed originally with spatial operators associated with the links of the tree. Each node with a label, or symbol name, represents an object in the image. The links connecting two nodes are signed with the relation operators. For the ordered subtree rooted at node S with n immediate descendants in the ordering $s_1, s_2, \ldots, s_n, S$ being the parent actually contains the *local body* consisting of all

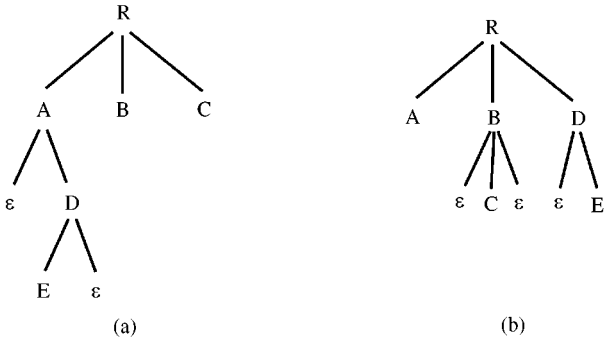


Figure 3. The general 2D C-trees of image f_2. (a) f_{2x} on x-coordinate axis; (b) f_{2y} on y-coordinate axis

its immediate child-nodes $s_1, s_2, \ldots, s_n$. For simplifying the tree representation, *empty-node* and *set-node* are employed in order to remove the relation operators from the tree according to the basic definition of the operators. An *empty-node* is a pseudo-node which is labeled 'ε' and is used to represent empty areas of any size. The relation operator of a link can be removed by inserting some suitable *empty-nodes*. Each *empty-node* is considered as an object of its own. The *set-node* is introduced for treating a set of lineage that each node has single-child-node. A *set-node* is a multi-label node consisting of objects that have the same begin-bound and end-bound. The detailed transformation rules are investigated by Hsu *et al.* [11]. The sample symbolic image f_2 in Figure 2(a) is represented in general 2D C-trees as shown in Figure 3.

Ordered labeled trees are trees whose nodes are labeled and in which the left to right ordering among siblings is significant. The distance and/or similarity of such trees have many applications in computer vision, pattern recognition, programming compilation and natural language processing [14]. The distance between two ordered trees is considered to be the weighted number of editing operations (insert, delete and relabel) to transform one tree to another. Each operation is associated with a cost function, denoted as Δ. Let E be a sequence $e_1, e_2, \ldots, e_k$ of editing operations. An E-derivation from tree $\mathfrak{I}_1$ to tree $\mathfrak{I}_2$ is a sequence of trees $A_0, A_1, \ldots, A_k$ such that $A = A_0$, $B = A_k$, and $A_{i-1} + A_i$, via e_i for $1 \leq i \leq k$. Then the cost of E-derivation is measured by the total weight of editing operations in E, i.e.

$$\Delta(E) = \sum_{i=1}^{|E|} \Delta(e_i) \tag{1}$$

Formally the editing distance between trees $\mathfrak{I}_1$ and $\mathfrak{I}_2$ is defined as

$$\delta(\mathfrak{I}_1, \mathfrak{I}_2) = \min\{\Delta(E) \mid E \text{ is an editing sequence from } \mathfrak{I}_1 \text{ to } \mathfrak{I}_2\} \tag{2}$$

In Hsu *et al.* [15], we proposed a 2D C-tree matching algorithm by modifying the tree matching algorithm developed by Zhang [16] to compute the editing distance between 2D C-trees for solving the image retrieval problem. Consider two pictures P_1 and P_2. Two 2D C-tree representations of P_1 (P_2), $\mathfrak{I}_{1x}(\mathfrak{I}_{2x})$ and $\mathfrak{I}_{1y}(\mathfrak{I}_{2y})$ along the x- and y-coordinate respectively, are constructed. We define the distance between P_1 and P_2,

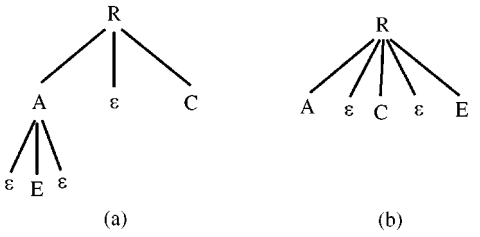


Figure 4. The 2D C-trees of query sketch q_2. (a) q_{2x} on the x-coordinate axis; (b) q_{2y} on the y-coordinate axis

$\delta(P_1, P_2)$, to be equal to $\delta(\mathfrak{I}_{1x}, \mathfrak{I}_{2x}) * \delta(\mathfrak{I}_{1y}, \mathfrak{I}_{2y})$. If one of the distances $\delta(\mathfrak{I}_{1x}, \mathfrak{I}_{2x})$ and $\delta(\mathfrak{I}_{1y}, \mathfrak{I}_{2y})$ is zero, $\delta(P_1, P_2)$ is defined to be equal to the other. The example picture f_2 in Figure 2(a) and query sketch q_2 in Figure 2(b) are used to illustrate the computation of picture distance. The 2D C-trees of f_2 and q_2 along the x-coordinate axis are in Figures 3(a) and 4(a), respectively. The editing distance along the x-coordinate axis between these two trees is the cost of editing operations needed to transform from f_{2x} to q_{2x}. At least two editing operations are needed. That is, in f_{2x} symbol D is deleted and symbol B is replaced by an empty node ε. The tree distance of $\delta(f_{2x}, q_{2x})$ is 2. Along the y-coordinate four delete operations are needed for symbols B, D and ε from f_{2y} to q_{2y}. That is, the cost of $\delta(f_{2y}, q_{2y})$ is 3. Finally, the distance of $\delta(f_2, q_2)$ is 6.

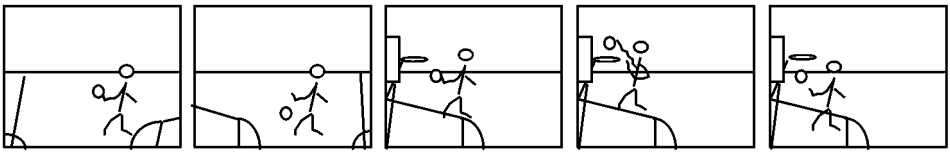
The proposed 2D C-tree matching algorithm takes $O(n_1^2 n_2^2)$ time complexity for computing the editing distance between two trees consisting of n_1 and n_2 nodes, respectively. While all the three distances between the query and the sample images in the database have been computed, the most similar image can be obtained. Suppose that there are N images in databases, $P_1, P_2, \ldots, P_N$, and a query image Q. The most similar image(s) to Q is

$$\{P_i \mid \delta(P_i, Q) \text{ is the minimum of } \delta(P_k, Q),\ 1 \le k \le N\}.$$

4. Video Information Retrieval

The structural matching of 2D C-trees plays an effective role for spatial query between images. Sometimes an image is viewed as a frame in video data ignoring sound. Then we can directly extend the approach of iconic image indexing and apply to video data indexing for video information retrieval.

A video sequence consists of a number of frames. Each frame can be constructed into two 2D C-trees along the x- and y-direction. The 2D C-trees preserve the spatial relationships among objects in an individual frame. A video sequence is then represented and indexed by an ordering set of 2D C-trees. The essence of video is that it captures the movements of objects, and it is therefore necessary to allow searching for a changing object relationship within a video sequence. The capability to find frames within a video sequence that contain objects in a specified relationship is useful [1]. Recall an example query in Section 1: *retrieve all the video sequences in which there is a running athlete holding a basketball, crossing over the field, jumping up, hanging over the basket and then slamming*. This query sequence can be expressed by five frames (Figure 5).



(a) The 1st frame (b) The 2nd frame (c) The 3rd frame (d) The 4th frame (e) The 5th frame

Figure 5. An example query sequence of five frames

For human beings, similarity retrieval is closer to conceptual thinking. Therefore, the query or 'match' need not be an 'exact match'. We match each query frame against each frame of every reference sequence in video databases. The matching process measures the similarity of frames, which computes the distance between the query frame and the reference frame. Note that if a reference frame r is matched by a query frame q, we need to count the distance between them, i.e. $\delta(r, q)$. Otherwise, we count the size of the frames r and q as the distance.

Definition 1 (*Size of a frame*). The size of a frame w is defined as the product of the sizes of its 2D C-tree representations along the x- and y-axis direction, w_x and w_y, respectively. The size of a 2D C-tree $w_x(w_y)$ is the number of symbols in $w_x(w_y)$, termed as $\eta(w_x)$ $(\eta(w_y))$, except the root node. Thus, the size of a frame w, referred to as $|w|$, is $(\eta(w_x)-1) * (\eta(w_y)-1)$.

For video sequence matching, we accumulate the total differences or distances between the frames of the query sequence and the frames of the reference sequence as the similarity of these sequences. The smaller the distance is, the more similar the two sequences are.

For the retrieval of similar video sequences, we propose three matching schemes in video information retrieval. For the illustration of these schemes, suppose there are m frames in the *reference sequence* V in the ordering of $v_1, v_2, \ldots, v_m$ and n frames in the *query sequence* U in the ordering of $u_1, u_2, \ldots, u_n$.

The full-sequence matching scheme is finding the minimum cost of full-sequence comparison between V and U. Every frame in the *reference sequence* and in the *query sequence* should be considered (see Figure 6).

The full-sequence comparison is defined by computing the editing distance required to change the *reference sequence* into the *query sequence*. Three types of changes between sequences are allowed: (1) delete—delete a frame from the *reference sequence*, (2) insert—insert a frame into the *query sequence*, and (3) replace—replace a frame of *reference sequence* with a frame of *query sequence*. For a change from $v_1, v_2, \ldots, v_i$ to $u_1, u_2, \ldots, u_j$, we compute the minimum cost of full-sequence comparison between V and U.

The second matching scheme is considered for query segment, called segment matching. Usually, a reference sequence contains lots of frames and users may not be able to specify an entire query sequence practically. Therefore, users specify an essential segment for video matching. For example, we may ask 'please retrieve video sequences that contain this query segment'. Consider the sequences V and U, and assume $m \gg n$.

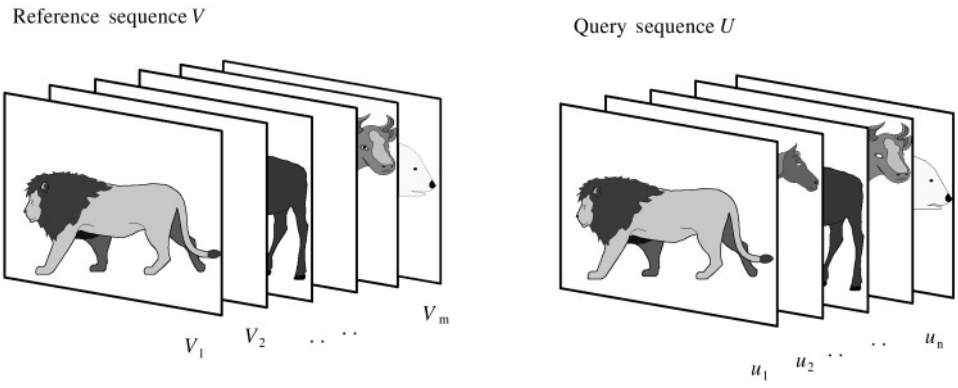


Figure 6. Full-sequence matching scheme

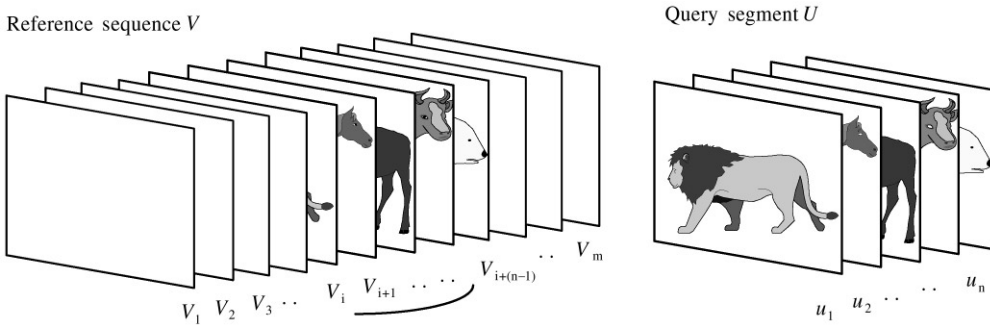


Figure 7. Segment matching scheme

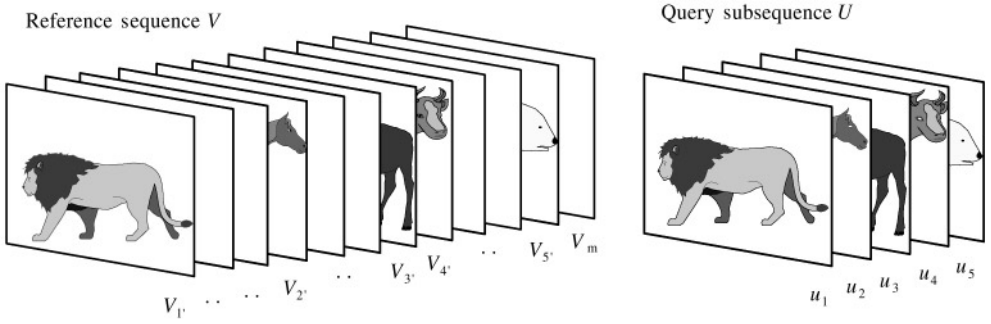


Figure 8. Subsequence matching scheme

We only compare the continuous n frames of the *reference sequence* V with the *query segment* U. A segment of V, $v_1 v_{i+1} \ldots v_{i+(n-1)}$, where $1 \le i \le m-n+1$, matches the query segment U, $u_1 u_2 \ldots u_n$, correspondingly (see Figure 7).

Sometimes users may specify a query via some key frames. These key frames may intermittently match a subsequence of the reference sequence. The third matching

scheme is used for matching the query subsequence, called subsequence matching. Only the distance of matched frames is considered. The unmatched frames being nonsignificant or redundant in reference sequence are ignored and their cost is not included. Figure 8 illustrates a subsequence matching between the *reference sequence* V and the *query subsequence* U. The frame of $V, v_{i'}$, matches the query frame u_i, $1 \le i \le n$. If $v_{i'}$ and $v_{j'}$ matches u_i and u_j correspondingly for $i<j$, then $i'<j'$ must hold.

Therefore, we can retrieve similar video sequences from the video databases via a certain query by using the above matching schemes.

5. Matching Algorithms

In this section we develop the matching algorithms for the video sequence matching schemes. For explaining the algorithm, the *reference sequence* $V = v_1 v_2 \ldots v_m$ and the *query* $U = u_1 u_2 \ldots u_n$ are used to illustrate the matching.

5.1. Full-Sequence Matching Scheme

The full-sequence matching between V and U is to compute the editing distance required to transform from V to U. Three types of editing operations between sequences are considered: delete, insert and replace. The cost of deleting a reference frame v_i or inserting a query frame u_j is weighted the size of the concerned frame, i.e. $|v_i|$ or $|u_j|$, respectively. The cost of the replacing operation is the distance between two transformed frames. For example, the ith frame of *reference* V, v_i, is replaced by the jth frame of *query* U, u_j. The replace cost is $\delta(v_i, u_j)$. For a transformation from $v_1, v_2 \ldots v_i$ to $u_1 u_2 \ldots u_j$, the minimum cost, denoted by $D[i, j]$ is defined as the minimum value of the following three cases.

(1) $D[i-1, j]+|v_i|$ (delete a frame v_i),
(2) $D[i, j-1]+|u_j|$ (insert a frame u_j),
(3) $D[i-1, j-1]+\delta(v_i, u_j)$ (replace frame v_i by frame u_j).

That is,

$$D[i,j] = \min\{D[i-1,j]+|v_i|,\ D[i,j-1]+|u_j|,\ D[i-1,j-1]+\delta(v_i,u_j)\}. \quad (3)$$

The well-known Symbolic String Sequence Comparison algorithm using the dynamic programming technique [17] can be extended to solve the frame sequence matching problem. Suppose that all the frames of the video sequence $V(U)$ are constructed into respective 2D C-tree representations, and the video sequence is represented and indexed by an ordering set of 2D C-trees. First, we must compute all the frame sizes and frame distances between those frames of V and U. Then the minimum change from V to U, denoted by $\Psi(V, U)$, is $D[m, n]$ computed by the above recurrence relation between $v_1 v_2 \ldots v_m$ and $u_1 u_2 \ldots u_n$, i.e. $\Psi(V, U) + D[m, n]$.

This Full-Sequence Matching algorithm is developed as follows.

Algorithm 1: Full-Sequence Matching Algorithm.
Input: The frame-to-frame distance table $\delta(v_i, u_j)$ for replacing v_i with u_j, and the frame size arrays $V[m]$, $U[n]$.

Output: The minimum distance between V and U, $\Psi(V, U)$.

```
Begin
  D [0,0] = 0;
  for i: =1 to m do D[i,0]=D[i−1,0]+V[i];
  for j: =1 to n do D[0,j]=D[0,j−1]+U[ j ];
  for i: =1 to m
      for j:=1 to n
          D[i, j]=min(D[i−1, j]+V[i], D[i, j−1]+U[ j ],
                  D[i−1, j−1]+δ(v_i, u_j));
  return D[m, n];
End;
```

It is clear from the above algorithm that the running time is $O(mn)$. An example is demonstrated below. Sequence V has six frames and sequence U has five frames. Suppose that the sizes of these 11 frames are all 25. The frame distances among them computed as described in Section 3 are made and listed in Table 2.

The computation of $D[6, 5]$ between sequences V and U is shown in Table 3. The minimum cost of editing distance between sequences V and U, $\Psi(V, U)$, is 36. From Tables 2 and 3 we can trace back the correspondence of matching frames. The correspondences $(v_1, u_1, 2)$, $(v_2, u_2, 2)$, $(v_3,$ null, $25)$, $(v_4, u_3, 6)$, $(v_5, u_4, 0)$, and $(v_6, u_5, 1)$ constitute an optimal matching [18]. Each triple $(v_i, u_j,$ cost) denotes a mapping from v_i to u_j associated with its cost between v_i and u_j. The notations we used in our sequence matching algorithm are: a delete operation when u_j is null and an insert operation if v_i is null. In this case, v_3 is deleted in the transformation from V to U.

Table 2. The frame distances between sequences V and U

$\delta(v_i, u_j)$	v_1	v_2	v_3	v_4	v_5	v_6
u_1	2	4	6	9	12	16
u_2	0	2	4	6	6	8
u_3	16	12	8	6	4	2
u_4	6	4	2	1	0	1
u_5	12	9	6	4	2	1

Table 3. The computation of D [6, 5] between sequences V and U

		V						
	D	0	1	2	3	4	5	6
	0	0	25	50	75	100	125	150
	1	25	2	27	52	77	102	127
U	2	50	25	4	29	54	79	104
	3	75	50	29	12	35	58	81
	4	100	75	54	31	13	35	59
	5	125	100	79	56	35	15	36

This minimum cost computation between two sequences is useful for retrieving similar sequences. Like the image query, we can retrieve similar sequences from databases via a query sequence based on editing distance. Suppose that there are N sequences in database, $S_1, S_2, \ldots, S_N$, and a query sequence T. The most similar sequence(s) to T is S_i, such that $\Psi(S_i, T)$ is the minimum one of all N sequences:

$$\{S_i \mid \Psi(S_i, T) \text{ is the minimum of } \Psi(S_k, T),\ 1 \le k \le N\}.$$

5.2. Segment Matching

Consider sequences V and U. We want to find a segment of V, $v_i v_{i+1} \ldots v_{i+(n-1)}$, where $1 \le i \le m-n+1$, which matches the query segment U correspondingly. Usually, the user cannot precisely specify the query segment frame by frame. We use a threshold τ to allow a certain difference between the corresponding frames. If the distance between two corresponding frames v_i and u_j is smaller than τ, these two frames match and constitute a matched pair (v_i, u_j). The problem becomes a string matching problem, which can be solved by the famous KMP algorithm [19] easily, if $\tau = 0$. Unfortunately, the tolerance τ somewhat affects the matching and the computation of the slide position for backtracking in the KMP algorithm. The slide operation must be rechecked and cannot be allowed when the distance is over the threshold. For such a reason, we must propose a segment matching algorithm:

Algorithm 2: Segment Matching Algorithm
Input: The frame-to-frame distance table $\delta(v_i, u_j)$ for replacing v_i by u_j, and a threshold τ.
Output: The start positions of matched subsequences in the sample video sequence.

```
Begin
  for i:=1 to m-n+1
    k=i; j=1;
    while j ≤ n then
    if δ(v_k, u_j)<τ then
        k=k+1; j=j+1;
    else
        break;
  if j>n then print matched position i;
End;
```

We use an example data extended from Table 2 to illustrate the segment matching process. Suppose that there are 12 frames in reference sequence V, and a query segment U consists of five frames. Their frame-to-frame distances are shown in Table 4.

Obviously, there does not exist any segment of five frames in sequence V matching the query segment U exactly. In other words, U is not an exact segment of sequence V. If we allow a threshold of $\tau = 10$, meaning that two frames match when their distance is smaller than 10, then we can obtain four segments of V matching the query, $u_1u_2u_3u_4u_5$, according to the above algorithm. That is $v_5v_6v_7v_8v_9$, $v_6v_7v_8v_9v_{10}$, $v_7v_8v_9v_{10}v_{11}$ and $v_8v_9v_{10}v_{11}v_{12}$. The costs of these matched segments are 15, 15, 17 and 20, respectively.

Table 4. The frame distances between reference sequence V and a segment U

$\delta(v_i, u_j)$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
u_1	4	2	0	1	2	4	6	9	12	16	16	20
u_2	9	4	2	1	0	2	4	6	6	8	12	16
u_3	20	20	20	16	16	12	8	6	4	2	1	0
u_4	20	16	12	8	6	4	2	1	0	1	2	4
u_5	20	20	16	16	12	9	6	4	2	1	0	1

It is interesting to find out that the cost of an unmatched segment $v_4v_5v_6v_7v_8$ is 19, which is in fact smaller than that of a matched subsequence $v_8v_9v_{10}v_{11}v_{12}$, which is 20. Thus, for different applications users may change the selecting criteria to be the total cost of two subsequences. We can make a little modification from the segment matching algorithm for this purpose.

Algorithm 3: Modified Segment Matching Algorithm

Input: The frame-to-frame distance table $\delta(v_i, u_j)$ for replacing v_i with u_j, and a threshold of total cost, ϖ.

Output: The start positions of matched subsequences in the sample video sequence.

```
Begin
  for i := 1 to m − n + 1
    k = i; j = 1; tc = 0;
    while j ≤ n then
      if (tc + δ(v_k, u_j)) < ϖ then
            k = k + 1; j = j + 1; tc = tc + δ(v_k, u_j);
      else
            break;
    if j > n then print matched position i;
End;
```

Note that these two segment matching algorithms also take $O(mn)$ time to match the segments.

5.3. Subsequence matching

The subsequence matching only considers the subsequence consisting of key frames specified in the query segment. The subsequence matching algorithm can employ the dynamic programming technique like the full-sequence matching algorithm (Algorithm 1) except ignoring the cost of unmatching frames in the reference sequence. In other words, the cost of deleting an unmatched frame in the reference sequence is weighted zero. On the contrary, every frame in a query segment needs to be matched. If a frame of a query segment is matched, the cost of matching is the distance between the corresponding frames. Otherwise, the cost of an unmatched frame in a query segment is measured in frame size as defined before. As stated above, the subsequence matching algorithm is developed as the following.

Algorithm 4: Subsequence Matching Algorithm

Input: The frame-to-frame distance table $\delta(v_i, u_j)$ for replacing v_i with u_j, and the query frame size $U[n]$.

Output: The minimum subsequence distance between V and U, $\Psi(V, U)$.

```
Begin
  for i := 0 to m do d[i, 0] = 0;
  for j := 1 to n do d[0, j] = d[0, j−1] + U[ j];
  for i := 1 to m
      for j := 1 to n
          d[i, j] = min(d[i−1, j], d[i, j−1] + U[j], d[i−1, j−1] + δ(v_i, u_j));
  return d[m, n];
End;
```

Obviously, the subsequence matching algorithm employing the dynamic programming technique also takes $O(mn)$ time complexity for computing the minimum subsequence distance.

We still use the example data in Table 4 to illustrate the subsequence matching. The computation of $d[m, n]$ between a sequence V and a query segment U is shown in Table 5.

It is assumed that the sizes of all the frames in query V are 25. From the computation of $d[i, j]$, we can easily trace back the optimal correspondence of subsequence matching. In this example, there are two corresponding mappings, referred to as the set of triples (v_i, u_j, cost), producing an optimal match in distance 5.

(1) $(v_3, u_1, 0)$, $(v_5, u_2, 0)$, $(v_9, u_3, 4)$, $(v_{10}, u_4, 1)$, and $(v_{11}, u_5, 0)$.
(2) $(v_3, u_1, 0)$, $(v_5, u_2, 0)$, $(v_{10}, u_3, 2)$, $(v_{11}, u_4, 2)$, and $(v_{12}, u_5, 1)$.

This means that the optimal subsequence matching between V and U is obtained at $\Psi(V, U) = 5$. Similarly, the subsequence matching approach provides a valuable solution for video sequence matching. Like the full-sequence, we can retrieve similar sequences from the sequences $S_1, S_2, \ldots, S_N$ via a certain query segment T specifying some key frames (see Figure 9). We only replace the full-sequence distance function $\Psi(S_i, T)$ with the subsequence distance $\psi(S_i, T)$. The most similar sequence(s) to T is

$$\{S_i \mid \psi(S_i, T) \text{ is the maximum of } \psi(S_k, T),\ 1 \le k \le N\}.$$

Table 5. The computation of $d[12, 5]$ between sequence V and a segment U

		V												
	d	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	25	4	2	0	0	0	0	0	0	0	0	0	0
U	2	50	29	8	4	1	0	0	0	0	0	0	0	0
	3	75	54	33	28	20	17	12	8	6	4	2	1	0
	4	100	79	58	45	36	26	21	14	9	6	5	4	4
	5	125	104	83	70	61	48	35	27	18	11	7	5	5

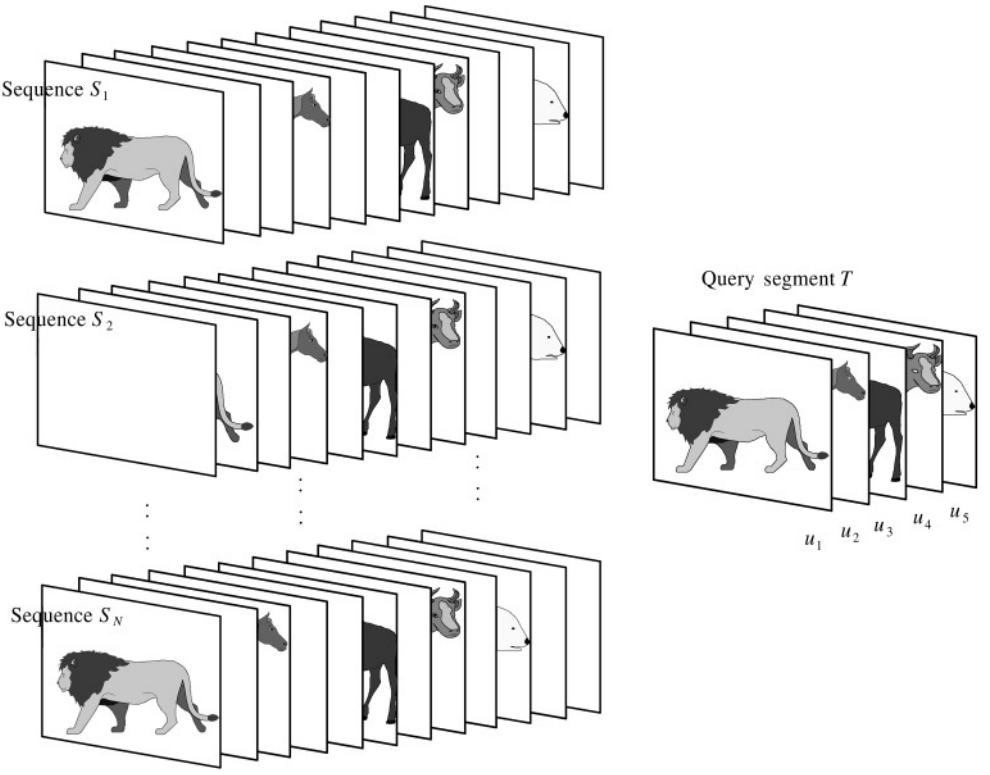


Figure 9. Choose the most similar sequence form $S_1, \ldots, S_N$ matching T

6. Approximate Sequence Matching

In general, similarity retrieval [20] is required since users usually cannot express queries in a precise way. For a frame of abundant contents, users often cannot specify a precise description in a query. The objects existing in a reference frame may not be expressed in a rough query. This strategy provides a more convenient and user-friendly manner in an interactive environment. The users can specify a query sequence consisting of some key frames concentrating on key objects. Only the key objects in the query sequence are considered and compared. In such a case, the superfluous objects in the query sequence are considered and compared. In such a case, the superfluous objects in a *reference* frame can be ignored on purpose when they do not appear in the *query* frame. This means that the cost of deleting an object in a *reference* frame is weighted zero. Meanwhile, the case of a relabel operation, which replaces a symbol in a *reference* frame with an empty-node in a *query* frame, is viewed as a special case of the delete operation. On the contrary, all the objects specified in a *query* are needed for measuring the distance. Therefore, the computation of editing distance between a *reference* frame and a *query* frame is modified into a partial function. The partial tree matching algorithm is proposed in Hsu *et al.* [15] for computing the partial tree distance. Note that this algorithm also takes $O(n_1^2 n_2^2)$ time complexity to compute the partial distance between two trees consisting of nodes n_1 and

n_2, respectively. We use γ(*reference*, *query*) to represent the partial distance between a *reference* frame and a *query* frame. Similar to the definition of tree distance δ, the partial distance γ is the product of two partial tree distances in two axis directions respectively.

To demonstrate the computation of partial distance, we use the example picture f_2 in Figure 2(a), which is regarded as the *reference* frame, and query sketch q_2 in Figure 2(b), which is considered as *query* frame. The 2D C-trees of f_2 and q_2 are in Figures 3 and 4, respectively. Because the editing operations needed to transform from f_2 to q_2 are all delete operations (even if a relabel operation needed in the x-direction is actually a special case of the delete operation) the partial tree distance between them are both zero for two axis directions, respectively. That is, the partial distance between f_2 and q_2, $\gamma(f_2, q_2)$, is zero. In such a case, the *reference* f_2 matches the *query* q_2 with zero cost. In other words, the *query* q_2 is a partial description of the *reference* f_2 without any difference.

Now, we can employ the similarity strategy via partial distance to integrate the matching algorithms discussed in Section 5. The example data in Table 6 is made for demonstrating the computation of *partial* distances between a *reference* sequence V and a *query* sequence U.

For the three matching schemes discussed in Section 4, the proposed matching algorithms can be modified and extended for approximate sequence matching by replacing $\delta(v_i, u_j)$ with the partial distance $\gamma(v_i, u_j)$. We use Algorithms 2 and 4 to illustrate the approximate sequence matching.

The Segment Matching algorithm can select the similar segment(s) in V under a threshold. We just replace $\delta(v_i, u_j)$ with $\gamma(v_i, u_j)$ in Algorithm 2. There are only two segments in V matched under $\tau = 3$, $v_6v_7v_8v_9v_{10}$ and $v_7v_8v_9v_{10}v_{11}$. The segment $v_6v_7v_8v_9v_{10}$ has the minimum cost correspondence for the query U.

The Subsequence Matching algorithm is modified similarly. The distance computation function $\delta(v_i, u_j)$ is replaced with the partial distance $\gamma(v_i, u_j)$. We can easily obtain an approximate subsequence in reference sequence V comparing with U. Only one matching correspondence, $\{(v_3, u_1, 0), (v_5, u_2, 0), (v_8, u_3, 2), (v_9, u_4, 0), (v_{11}, u_5, 0)\}$, is selected under a constraint threshold 4, representing the size of each query frame. The computation of approximate sequence matching is shown in Table 7 and the optimal matching is obtained at a cost equal to 2.

Therefore, the approximate sequence matching can provide a comprehensive retrieval of video data. The general mechanism in retrieving the most similar sequence(s) from N video sequences in the database, $S_1, S_2, \ldots, S_N$, via a query sequence T is performed through the following three steps.

Table 6. The partial distances between source stream V and a sequence U

$\gamma(v_i, u_j)$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
u_1	2	1	0	1	1	1	2	3	3	4	4	4
u_2	3	2	1	1	0	1	2	2	2	3	3	4
u_3	4	4	4	4	4	3	3	2	2	1	1	0
u_4	4	4	3	2	2	2	2	1	0	1	1	2
u_5	4	4	4	4	3	2	2	2	1	1	0	1

Table 7. The computation of approximate sequence matching between V and U

		V												
	d	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	4	2	1	0	0	0	0	0	0	0	0	0	0
U	2	8	6	4	2	1	0	0	0	0	0	0	0	0
	3	12	10	8	6	5	4	3	3	2	2	1	1	0
	4	16	14	12	10	8	7	6	5	4	2	2	2	2
	5	20	18	16	14	12	11	9	8	7	5	3	2	2

Step 1: Compute all the *partial* distances of the frame pairs between a sequence S_k and T, $\gamma(s_{ki}, t_j)$, $1 \le i \le |S_k|$, $1 \le j \le |T|$.

Step 2: Compute the minimum partial subsequence matching for S_k and T, $\psi(S_k, T)$.

Step 3: Repeat Steps 1 and 2 for all sequence S_k, $1 \le k \le N$. Then choose the one(s) with minimum cost to be the most similar matched video sequence(s) for T.

In summary, for a query sequence T consisting of n frames, we must compute the minimum partial distance $O(mn)$ time taken to compare N reference sequences that have m frames and choose the most similar sequence with minimum cost in the databases.

7. Prototype System

We use the above mechanisms to implement an Interactive Video Information System (IVIS) in our experimental project. The schematic diagram of the IVIS is shown in Figure 10.

There are three subsystems in the IVIS. The query subsystem provides users an interactive interface to query and browse the retrieved results. The system supports single frame query and frame sequence query. The interface allows users to draw a query image by assembling object icons designed to use an existing query template consisting of query frames. The subsystem contains a query processing module and an interactive browser module. To achieve content-based retrieval, a friendly user interface is required for users to refine query visually and navigate their way through the database visually.

The information management subsystem is the core of an IVIS. There are four modules in the subsystem: content acquisition module, video indexing module, video store/retrieval module and index matching module. When we collect the source video streams, the content acquisition module processes the raw data and identifies some key image frames. The representative objects in key frames are also extracted. These tasks are not easy to achieve automatically using techniques in pattern recognition and image understanding at the current stage and are performed instead in a human-assisted fashion. After extracting the representative objects from identified key frames, the video indexing module can construct the 2D C-trees to be the representative indexing of key frames. All the video streams and key frames are stored in the data storage subsystem by the video store/retrieval module. The index matching module computes the minimum

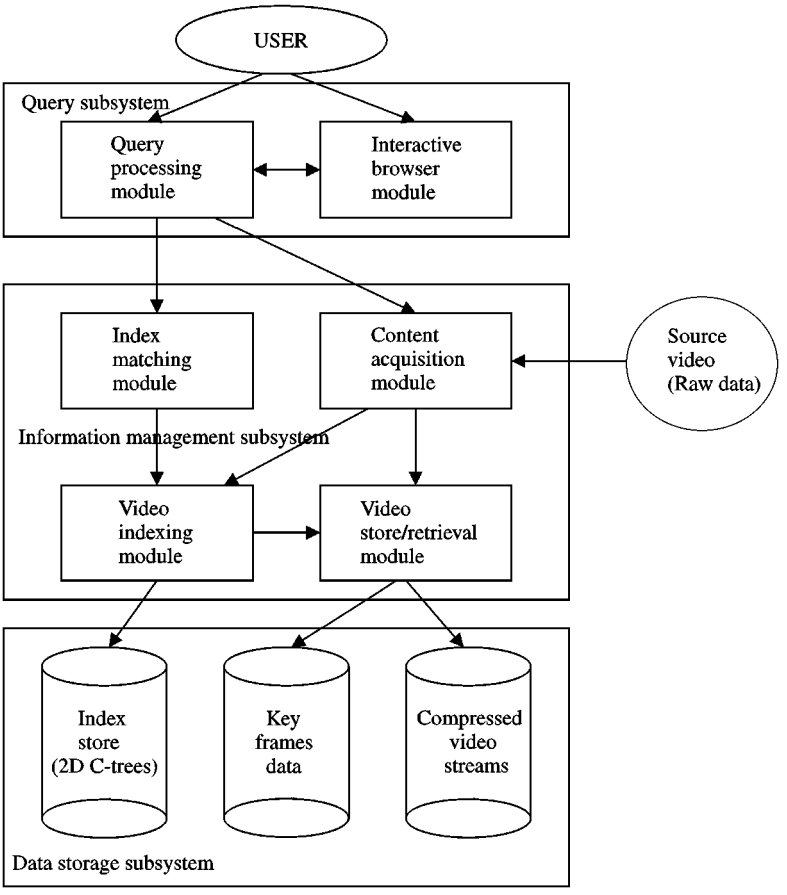


Figure 10. Schematic diagram of an interactive video information system

matching distance for the query processing subsystem based upon the proposed approximate sequence matching mechanism.

There are three major databases in the data storage subsystem. The constructed 2D C-trees are the representative indexing of video streams and are kept in the Index Store. All the source video data in the AVI file format are compressed to the compressed video streams for reducing the storage space. The identified key frame data are also collected in the database for providing the annotation with indices to speed up retrieval of the desired video streams.

In the experimental project, we capture 48 streams from 'The Lion King' cartoon produced by The Walt Disney Company. Each stream takes 99 s and consists of about 1500 frames. There are 351 key frames that have been identified in all and some are listed in Appendix A. Each stream has 7 or 8 key frames on an average. For this popular animation, 78 roles are chosen to be the objects, which are extracted from the key frames in a human-assisted fashion. These objects are classified into 28 categories represented by a set of designed icons in the system. The labels of objects are listed in Appendix B. The objects and their bounding rectangles within images are also extracted

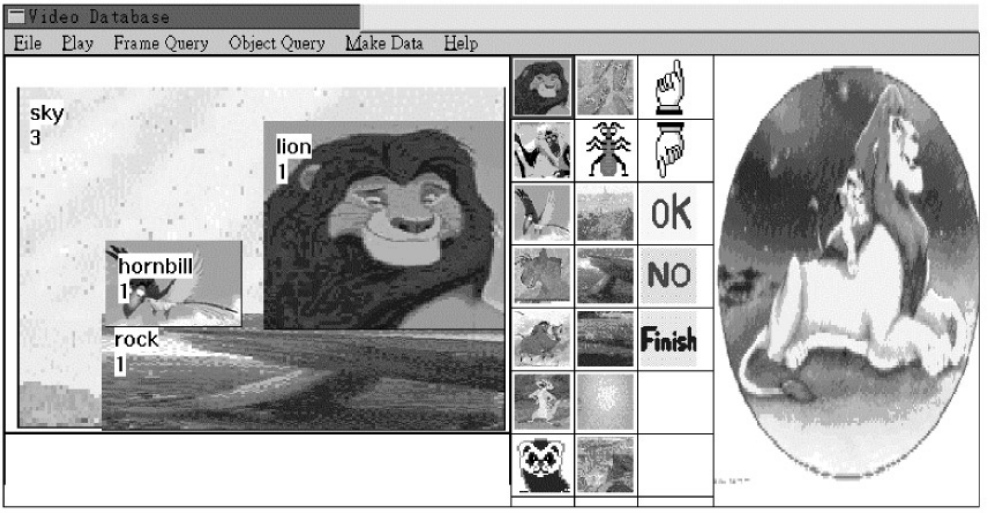


Figure 11. An example template query for approximate sequence matching

after capturing the images from the source video. Each frame containing about five objects on an average is constructed into two 2D C-trees along the *x*- and *y*-axis directions independently and these two indexing representations have been stored. The number of objects within each frame implies the number of nodes in its corresponding 2D C-trees.

In IVIS, we concentrate on video information retrieval based upon approximate sequence matching mechanism. We allow users to draw a sequence of query frames by assembling object icons designed or use an existing query template consisting of some consecutive query frames. An example is shown in Figure 11. Before computing the distances among frames, two 2D C-trees of each frame in the *query* sequence are constructed first. Starting from the first *reference* sequence of the video database, we compute all the *partial* distances between the frame pairs of this *reference* sequence and the *query* sequence. The subsequence matching distance between this *reference* sequence and the *query* sequence is easily computed by Algorithm 4. While all the sequences of the database are compared with the query one by one, the minimum one represents the most similar sequence for the query. The result via a query template in Figure 11 is shown in Figure 12. After retrieving a stream, the user can manipulate the stream via basic operations, such as *Play*, *Reverse*, *Pause*, *Goto*, *Backward*, *Forward* and so on.

For the 48 video streams, we have used each representative key frame sequence to be the test query sequences against the others. Each comparison takes 15 to 25 ms in a Pentium-133 CPU with 64MB RAM environment. It is clear that all the video streams are exactly retrieved from the databases without distance zero. Our initial results validate the effectiveness of similarity retrieval by 2D C-trees matching.

8. Conclusions

Chang *et al.* proposed an effective 2D string approach for spatial indexing of image data. To represent the spatio-temporal information, we extend the approach of iconic image

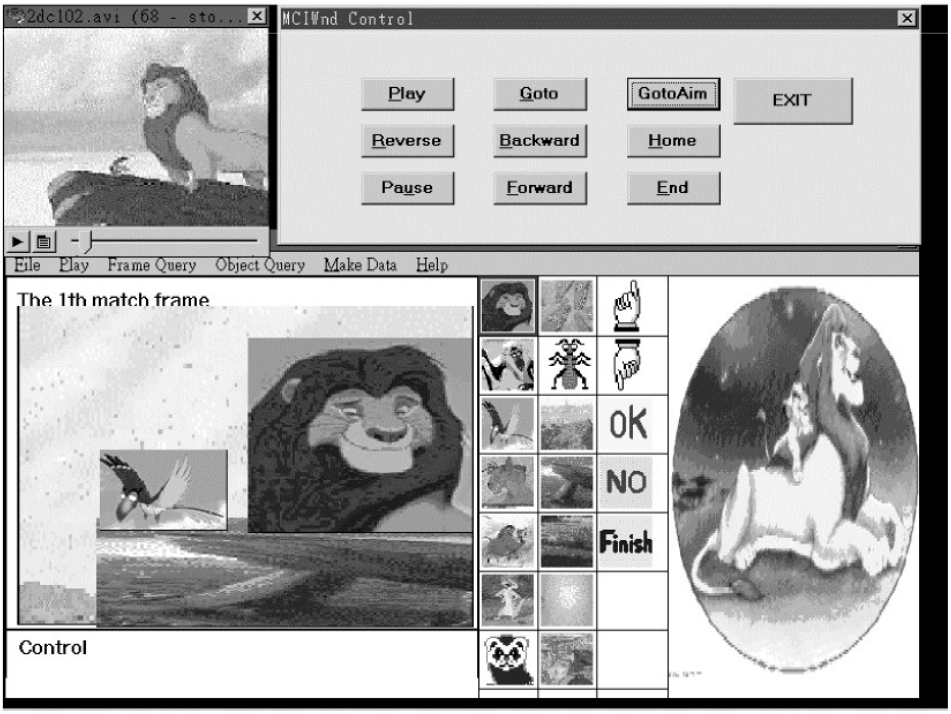


Figure 12. A result stream via a query template in Figure 11

indexing to video data indexing in this paper. We investigate a new knowledge model, 2D C-tree, to characterize the spatial information of video content. The indexing technique represents the spatial content within individual frames and constructs an ordering set of 2D C-trees for a video sequence. Each frame is constructed into two representative 2D C-trees along the *x*- and *y*-axes. The ordering set of 2D C-trees then becomes the representative indexing of the video sequence. The similarity between two frames is measured by the editing distance between their corresponding 2D C-trees. The video sequence matching problem is solved by computing the minimum cost of corresponding frames. For various applications or demands, we propose three matching schemes for video information retrieval: *full-sequence matching, segment matching and subsequence matching*. We also extend the matching schemes to approximate sequence matching by computing the partial distances between *reference* frames and *query* frames. The approximate sequence matching provides a comprehensive operation for retrieving video data. We also briefly present our developed prototype based on the proposed strategy in video information retrieval.

This work concentrates on handling the spatial-temporal relationships of video data. Nevertheless, how to integrate with other useful features [21], such as color, shape, texture, and even voice, for video retrieval is worthy of exploring in many practical applications, such as news database, electronic shopping and distance learning. Moreover, the automatic content extraction, especially object recognition or identification in video frames, is also among our future research direction.

Acknowledgement

We are grateful to The Walt Disney Company for allowing us the experimental use of 'The Lion King' production.

Appendix A

Some key frames of the 48 streams collected in our project

Appendix B

Seventy eight roles are chosen to be the objects in 'The Lion King' cartoon

Category	Object
Lion	Mufasa, Simba_cub, Simba_adult, Scar, Simba_son, Sarabi, Nana_cub, Nana_adult, Sanafina, Lioness, Lionesses
Baboon	Rafiki, Baboon1, Baboons
Hornbill	Zazu
Hyena	Banzai, Shenzi, Ed, Hyena1, Hyenas
Warthog	Pumbaa
Meerkat	Timon
Animal	Hippo, Elephant, Rhino, Giraffe, Zebra, Antelope, Leopard, Ape, Vulture, Wildebeest, Squirrel, Bird, Crocodile, Mouse
Animals	Hippos, Elephants, Rhinos, Giraffes, Zebras, Antelopes, Leopards, Apes, Vultures, Wildebeests, Squirrels, Birds, Crocodiles
Insect	Ant, Butterfly, Worm
Land	Pride_land, Cracked_land
Rock	Pride_rock, Rock
Grass	Dark_grass, Bright_grass
Sun	Sun
Valley	Valley
Tree	Tree1, Trees, Forest
Sky	Dark_sky, With_stars, Bright_sky
Termite_mt	Termite_mt
Gorge	Gorge
Thornbushes	Thornbushes
Flame	Flame
Jungle	Jungle
Pond	Pond
Waterfall	Waterfall
Stream	Stream
Mount	Mount
Flowers	Flowers
Skull	Skull
Cub_painting	Cub_painting

References

1. K. Shearer, S. Venkatesh & D. Kieronska (1996) Spatial indexing for video databases. *Journal of Visual Communication and Image Representation* 7, 325–335.
2. E. Oomoto & K. Tanaka (1993) OVID: Design and implementation of a video-object database system. *IEEE Transactions on Knowledge and Data Engineering* 5, 629–643.
3. S. W. Smoliar & H. Zhang (1994) Content-based video indexing and retrieval. *IEEE Multimedia* 1, 62–72.

4. C. W. Chang (1996) *Structured video computing and content-based retrieval in a video information system*. Ph.D dessertation, National Chiao Tung University, Taiwan, R.O.C.
5. A. D. Narasimhalu (1995) Special section on content-based retrieval. *ACM Multimedia Systems* 3, 1–2.
6. T. S. Chua & L. Q. Ruan (1995) A video retrieval and sequencing system. *ACM Transactions on Information systems* 13, 373–407.
7. J. K. Wu, A. D. Narasimhalu, B. M. Mehtre, C. P. Lam & Y. J. Gao (1995) CORE: a content-based retrieval engine for multimedia information systems. *ACM Multimedia Systems* 3, 25–41.
8. A. Gupta, T. Weymouth & R. Jain (1991) Semantic queries with pictures: the VIMSYS model. In: *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, pp. 69–79.
9. S. K. Chang, Q. Y. Shi & C. W. Yan (1987) Iconic indexing by 2-D strings. *IEEE Transaction on Pattern Analysis and Machine Intelligent* PAMI-9, 413–428.
10. S. Y. Lee, M. K. Shan & W. P. Yang (1989) Similarity retrieval of iconic image database. *Pattern Recognition* 22, 675–682.
11. F. J. Hsu, S. Y. Lee & P. S. Lin (1997) 2D C-tree spatial representation for iconic image. In: *Proceedings of the 2nd International Conference on Visual Information Systems* (*Visual'97*), San Diego, CA, pp. 287–294.
12. S. Y. Lee & F. J. Hsu (1990) 2D C-string: a new spatial knowledge representation for image database systems. *Pattern Recognition* 23, 1077–1087.
13. S. Y. Lee & F. J. Hsu (1992) Spatial reasoning and similarity retrieval of images using 2D C-string knowledge representation. *Pattern Recognition* 25, 305–318.
14. K. S. Fu (1982) *Syntactic Pattern Recognition and Application*. Prentice-Hall, Englewood Cliffs, NJ.
15. F. J. Hsu, S. Y. Lee & P. S. Lin (1998) Similarity retrieval by 2D C-trees matching in image databases. *Journal of Visual Communication and Image Representation* 9, 87–100.
16. K. Zhang (1995) Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern Recognition* 28, 465–474.
17. U. Manber (1989) *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, Reading, MA.
18. Y. P. Wang & T. Pavlidis (1990) Optimal correspondence of string subsequences. *IEEE Transaction on Pattern Analysis and Machine Intelligent* PAMI-12, 1080–1087.
19. D. E. Knuth, J. H. Morris & V. R. Pratt (1977) Fast Pattern matching in strings. *SIAM Journal of Computing* 6, 323–350.
20. P. Ciaccia, F. Rabitti & P. Zezula (1996) Similarity Search in Multimedia Database Systems. In: *Proceedings of the 1st International Conference on Visual Information Systems* (*Visual'96*), Melbourne, Australia, pp. 107–115.
21. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele & P. Yanker (1995) Query by image and video content: the QBIC system. *IEEE Computer* 44, 23–32.