PERGAMON

Pattern Recognition 35 (2002) 2521–2537

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

3D C-string: a new spatio-temporal knowledge representation for video database systems

Anthony J.T. Lee *, Han-Pang Chiu, Ping Yu

Department of Information Management, National Taiwan University, Taipei 106, Taiwan, R.O.C

Received 3 November 1999; received in revised form 17 September 2001; accepted 25 September 2001

Abstract

In video database systems, one of the most important methods for discriminating the videos is by using the objects and the perception of spatial and temporal relations that exist between objects in the desired videos. In this paper, we propose a new spatio-temporal knowledge representation called 3D C-string. The knowledge structure of 3D C-string, extended from the 2D C^+-string, uses the projections of objects to represent spatial and temporal relations between the objects in a video. Moreover, it can keep track of the motions and size changes of the objects in a video. The string generation and video reconstruction algorithms for the 3D C-string representation of video objects are also developed. By introducing the concept of the template objects and nearest former objects, the string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D C-string is unique too. This approach can provide us an easy and efficient way to retrieve, visualize and manipulate video objects in video database systems. Finally, some experiments are performed to show the performance of the proposed algorithms.

Keywords: Video database; Spatio-temporal relation; 2D-string; 2D C^+-string; 3D-string; 3D C-string

1. Introduction

In video database systems, one of the most important methods for discriminating the videos is by using the objects and the perception of spatial and temporal relations that exist between objects in the desired videos. Therefore, how videos are stored in a database becomes an important design issue of video database systems. The spatio-temporal knowledge embedded in videos should be preserved in the knowledge structure and the knowledge structure should be object-oriented, so that users can easily retrieve, visualize and manipulate videos in video database systems. In comparison with text, image, and audio, video contains richer information [1,2]. But the richness results in the lack of generally accepted representation of a video. Oomoto and Tanaka [3] considered a video as a sequence of video frames and represent a video as a collection of attribute/value pairs. Weiss et al. [4] used algebraic operators to assemble videos. Smoliar and Zhang [5] modeled the content of videos in two steps. First of all, videos were classified into classes and these classes formed a tree of topical categories according to their topics. By taking a horizontal or vertical slice on a video shot, the movement of a symbol in the video can be traced.

Chang et al. [6] proposed the concept of 2D string to represent the spatial relations between the objects in an image (image objects for short). The basic idea was to project the image objects onto the x- and y-axis to form two strings representing the relative positions of the projections in the x- and y-axis, respectively. The knowledge structure of 2D string [7] used the projections of image objects to represent spatial relations between the objects. An image query can also be specified as a 2D string. This approach provided a natural way to construct iconic indexes for images and supported spatial reasoning, image queries, visualization, and image manipulation. There was a lot of follow-up

* Corresponding author. Tel.: +886-2-2363-0231x2978; fax: +886-2-2363-1327.

E-mail address: jtlee@im.ntu.edu.tw (A.J.T. Lee).

0031-3203/02/$22.00
PII: S0031-3203(01)00224-2

research based on the concept of 2D string including 2D G-string [8], 2D C-string [9–11], 2D C^+-string [12], and 2D RS-string [13].

Liu and Chen [14] extended the notion of 2D string to meet the requirement and characteristics of videos and defined 3D string to represent the relations between the objects in a video (video objects for short). The knowledge structure of 3D string used the projections of video objects to represent spatial and temporal relations between them. The basic idea was to project the video objects onto the x-, y-, and time-axis to form three strings representing the relative positions of the projections in the x-, y-, and time-axis, respectively. This approach can provide an easy way to retrieve, visualize and manipulate objects in video database systems. But according to their definition, they only recorded the central point and starting frame number for a video object. So they cannot realize the spatial overlapping relations and precise temporal relations between the video objects. The information about the motions and size changes of video objects was omitted in their work.

Therefore, we need a more compact and precise knowledge structure to represent spatio-temporal relations between the video objects and to manipulate the information about the motions and size changes associated with them. In this paper, we extend the concepts of 2D C^+-string [12] to meet the requirement and characteristics of a video. We propose 3D C-string to represent the spatio-temporal relations between the video objects and to overcome the weakness of 3D string. 3D C-string can keep track of the motions and size changes associated with the video objects and represent more precise spatio-temporal relations between the video objects.

In Section 2, we give a brief analysis of previous approaches of 2D string and 3D string. Then we explain the reason why we extend the concepts of 2D C^+-string to overcome the weakness of 3D string. In Section 3, we present 3D C-string, a new spatio-temporal knowledge representation of a video. The string generation algorithm is described in Section 4. Then we propose a video reconstruction algorithm based on the 3D C-string representation in Section 5. In Section 6, the results on some performance experiments are presented. Finally, conclusions are made in Section 7.

2. Analysis of previous 2D and 3D string approaches

Chang et al. [6] proposed the 2D strings to represent spatial relations of image objects by their projections. An image objects is enclosed by a minimum bounding rectangle (MBR). The reference point of an image object is the centroid of its MBR. A symbolic picture is formed by collecting the MBRs of the image objects in the original image. The symbolic picture can be used to represent the spatial relations between the image objects and is encoded as a 2D string.

The advantage of this spatial knowledge representation is that 2D strings can be used in iconic indexing and spatial reasoning for image database systems. Since a symbolic picture can also be quickly reconstructed from such a 2D string for visualization, the 2D string representation with appropriate picture reconstruction algorithm can also be used for browsing the images in an image database. There are three spatial relation operators employed in 2D strings. The symbol "$<$" denotes the "left–right or below–above" spatial relation. The symbol "$=$" denotes the "at the same spatial location as" relation and the symbol ":" stands for "in the same set as" relation.

Later, Jungert and Chang [8,15,16] extended the idea of 2D strings to form 2D G-strings by introducing several new spatial operators to represent more topological relations between the image objects. The 2D G-string representation embeds more information about spatial relations between the image objects, hence facilitates spatial reasoning about shapes and relative positions of the image objects.

Following the same concept, Lee and Hsu [9] proposed the 2D C-string representation based on a special cutting mechanism. Since the number of subparts generated by this new cutting mechanism is reduced significantly, the string representing an image is much shorter while still preserving the spatial relations between the image objects. The 2D C-string representation is more economical in terms of storage space and navigation complexity in spatial reasoning. In 2D C-string, there are 13 types of spatial relations between two one-dimensional (1D) intervals. One of them is "equal" relation and six of them are symmetric relations of the others. Hence, those relations can be represented by seven spatial operators whose notations and conditions are listed in Table 1, where Begin(A) and End(A) are the begin-bound and end-bound of the x- (or y-) projection of object A, respectively.

The 2D C-string representation captures the spatial relations between the image objects; however, it ignores the information about relative sizes and locations of the image objects. Hence, the 2D C-string representation results in ambiguity of spatial relations between the image objects. The reason of producing the ambiguity is that the metric information is ignored in a 2D C-string.

To overcome this problem, Huang and Jean [12] proposed the knowledge structure 2D C^+-string. In 2D C^+-string, each

Table 1
The definitions of spatial operators in 2D C-string

Notations	Conditions
$A < B$	$\text{End}(A) < \text{Begin}(B)$
$A = B$	$\text{Begin}(A) = \text{Begin}(B)$, $\text{End}(A) = \text{End}(B)$
$A \mid B$	$\text{End}(A) = \text{Begin}(B)$
$A\%B$	$\text{Begin}(A) < \text{Begin}(B), \text{End}(A) > \text{End}(B)$
$A[B$	$\text{Begin}(A) = \text{Begin}(B)$, $\text{End}(A) > \text{End}(B)$
$A]B$	$\text{Begin}(A) < \text{Begin}(B)$, $\text{End}(A) = \text{End}(B)$
A/B	$\text{Begin}(A) < \text{Begin}(B) < \text{End}(A) < \text{End}(B)$

image object has two pairs of begin-bounds and end-bounds. One of them is for the x-projection of the image object and the other for the y-projection. From the begin-bounds and end-bounds of the projections, the sizes of image objects and the distances between image objects can be calculated. It only has to calculate three kinds of metric information: the size of image object $A = \mathrm{End}(A) - \mathrm{Begin}(A)$, operator "$<$" with the distance between image objects A and $B = \mathrm{Begin}(B) - \mathrm{End}(A)$, operator "%" with the distance between image objects A and $B = \mathrm{Begin}(B) - \mathrm{Begin}(A)$. Using the 2D C^+-string representation, the ambiguity problem in 2D C-strings is resolved.

In the knowledge structure of 3D string [14], video objects are projected onto the x-, y-, and time-axis to form three strings representing the relative positions and relations of the projections in the x-, y-, and time-axis, respectively. A video object is represented by its central point and starting frame number. Two operators "$|_n$" and "$\equiv$" are introduced in 3D string. The operator "$|_n$" denotes the distance between two video objects. $A\,|_n\,B$ denotes that video object A is adjacent to video object B and the distance between the central points of video objects A and B is n. "$\equiv$" denotes the appositional relation. $A \equiv B$ denotes that video object A is appositional to video object B. Since 3D string cannot represent the information about the motions and size changes of video objects, it results in a certain ambiguity. Let us consider the example shown in Fig. 1.

In Fig. 1, videos M and N appear obviously different. But they have the same 3D string. Their 3D X-string is $A \equiv B$, Y-string is $B\,|_3\,A$, time-string is $A \equiv B$. We can see that videos M and N have the same 3D X- and Y-strings although video objects A and B have different sizes, and spatial relations. This is because both video objects A in videos M and N have the same central point and both video objects B have the same central point too. Videos M and N have the same 3D time-string although video objects A and B have different motion and temporal relations. This is because they have the same starting frame. This example shows that 3D string cannot manipulate the information about the motions and size changes of video objects. It cannot represent spatial and temporal relations between the video objects precisely either.

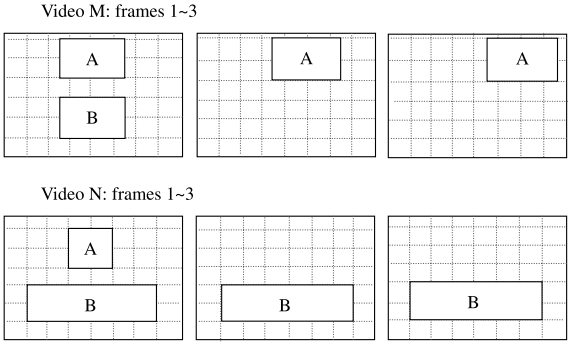


Fig. 1. Two different videos have the same 3D string.

Therefore, we need one more compact and precise knowledge structure to represent spatio-temporal relations between the video objects and to manipulate the information about the motions and size changes associated with the video objects.

Based on the above analysis, in this paper we propose a new spatio-temporal knowledge representation for videos, called 3D C-string. The 3D C-string is extended from the concepts of the 2D C^+-string, and it can overcome the weakness of 3D string. 3D C-string can keep track of the motions and size changes associated with video objects and preserve more precise spatio-temporal relations between the video objects.

3. 3D C-string representation of symbolic videos

In the knowledge structure of 3D C-string, video objects are projected onto the x-, y-, and time-axis to form three strings to represent the spatial (or temporal) relations between the projections in the x-, y-, and time-axis, respectively. In comparison with 2D C^+-string, 3D C-string has one more dimension: time dimension. So 3D C-string is different from 2D C^+-string that has only spatial relations between the image objects, it has spatial and temporal relations between the video objects. Hence, it is required to keep track of the information about the motions and size changes of the video objects in 3D C-string.

There are 13 relations for one-dimensional intervals in the knowledge structure of 3D C-string. For the x (or y) dimension, there are 13 spatial relations between the x- (or y-) projections of video objects and the corresponding spatial operators have been listed in 2D C-string [9] as shown in Table 1. In the time dimension, there are 13 temporal relations between the time-projections of video objects, too. So we use the same temporal operators as the spatial operators. For example, in the x (or y) dimension, $A < B$ represents that the x-projection of video object A is before that of video object B. In the time dimension, $A < B$ denotes that video object A disappears before video object B appears.

A video object in the knowledge structure of 3D C-string is approximated by a minimum bounding rectangle (MBR) whose sides are parallel to the x- or y-axis. For each video object, we keep track of the initial location and size of the video object. That is, we keep track of the location and size of a video object in its starting frame. After keeping track of the initial locations and sizes of video objects, we record the information about the motions and size changes of the video objects in the knowledge structure of 3D C-string.

To record the time points when the motion and size of a video object is changed, we introduce one more temporal operator, "$|_t$". Operator $|_t$ denotes that a video object may change its state, including the motion and size change of the

video object. For example, $A_3 |_t A_6$ denotes that in the first three frames, video object A remains in the same state of the motion and size change. However, from the fourth frame to the ninth frame, the state of the motion and size change of video object A is changed into another.

Based on the above analyses and discussions, we propose the knowledge structure of 3D C-string to represent spatio-temporal relations between the video objects of interest.

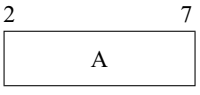
Definition 1. The knowledge structure of 3D C-string is a 7-tuple $(\mathscr{O}, \mathscr{A}, \mathscr{C}, \mathscr{R}_g, \mathscr{R}_l, \mathscr{R}_m, \text{“( )”})$ where

(1) $\mathscr{O}$ is the set of video objects of interest;
(2) $\mathscr{A}$ is the set of attributes to describe the objects in $\mathscr{O}$;
(3) $\mathscr{C}$ is the cutting mechanism, which consists of a set of cutting lines;
(4) $\mathscr{R}_g = \{\text{“<”}, \text{“|”}, \text{“}|_t\text{”}\}$ is the set of global relation operators, where “$|_t$” only uses in the time dimension. A “$|_t$” operator is generated for each time point when the state of a video object is changed;
(5) $\mathscr{R}_l = \{\text{“=”}, \text{“[”}, \text{“]”}, \text{“\%”}\}$ is the set of local relation operators;
(6) $\mathscr{R}_m = \{\text{“}\uparrow\text{”}, \text{“}\downarrow\text{”}\}$ is the set of motion operators to denote the direction of the motion of a video object, and it only uses in the u- and v-strings. Operator $\uparrow$ denotes that the video object moves along the positive direction of the x- (or y-) axis. Operator $\downarrow$ denotes that the video object moves along the negative direction of the x- or (y-) axis;
(7) “( )” is a pair of separators which are used to describe a set of video objects as a spatial template object or temporal template object.

According to the research result of Lee and Hsu [9], we know that all 13 operators except “/” can precisely represent the relations (no ambiguity) between two objects. To avoid using the “/” operator, we can use $A]B\,|\,B$ to replace A/B in our cutting mechanism and string generation algorithm described in Section 4.

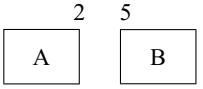
Next, we add some metric information to the knowledge structure of 3D C-string.

1. Video object A with size s of its x-projection is denoted as A_s, where $s = \text{End}_x(A) - \text{Begin}_x(A)$, where $\text{Begin}_x(A)$ and $\text{End}_x(A)$ are the begin-bound and end-bound of the x-projection of video object A, respectively. Video object A with size s' of its y-projection is denoted as $A_{s'}$, where $s' = \text{End}_y(A) - \text{Begin}_y(A)$. Similarly, video object A with size (or length) s'' of its time-projection is denoted as $A_{s''}$, where $A_{s''} = \text{End}_{\text{time}}(A) - \text{Begin}_{\text{time}}(A)$.
   For example,



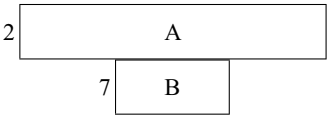
   is represented by A_5.

2. Operator “$<$” with the distance d between the x- (or y-) projection of video object A and that of video object B is denoted as $A <_d B$, where $d = \text{Begin}_x(B) - \text{End}_x(A)$ (or $d = \text{Begin}_y(B) - \text{End}_y(A)$). In the knowledge structure of 3D C-string, we only keep track of the initial location of a video object. Hence, for $A <_d B$, the starting frame of video object A may be different from that of video object B. Similarly, operator “$<$” with the distance d' between the time-projection of video object A and that of video object B is denoted as $A <_{d'} B$, where $d' = \text{Begin}_{\text{time}}(B) - \text{End}_{\text{time}}(A)$.
   For example,



   is represented by $A <_3 B$.

3. Operator “%” with the distance d between the x- (or y-) projection of video object A and that of video object B is denoted as $A\%_d B$, where $d = \text{Begin}_x(B) - \text{Begin}_x(A)$ (or $d = \text{Begin}_y(B) - \text{Begin}_y(A)$). Similarly, operator “%” with the distance d' between the time-projection of video object A and that of video object B is denoted as $A\%_{d'} B$, where $d' = \text{Begin}_{\text{time}}(B) - \text{Begin}_{\text{time}}(A)$.
   For example,



   is represented by $A\%_5 B$.

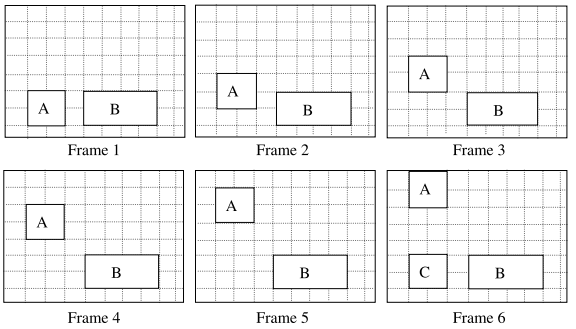
4. Operators “$\uparrow_{v,r}$” and “$\downarrow_{v,r}$” have two subscripts (fields). v is the velocity of the motion and r is the rate of size change of the video object. For example, u-string $A \uparrow_{2,1}$ denotes that video object A moves along the positive direction of the x-axis with the velocity $= 2$ and the rate of size change $= 1$. That is, the size of video object A remains unchanged.
5. Other operators: no metric information.

To see how 3D C-string works, let us consider the following example as shown in Fig. 2. The projections of the initial locations of video objects A, B, and C are shown in Fig. 3. The corresponding 3D C-string of the video is shown in Fig. 2(b).

From frame 1 to 6, video object A moves from bottom to top along the positive direction of the y-axis with the velocity of 1 unit/frame, but no motion along the x-axis. So A_2 is followed by operator $\uparrow_{0,1}$ in the u-string and followed by operator $\uparrow_{1,1}$ in the v-string.

If we add three more frames to the previous video as shown in Fig. 4(a), the corresponding 3D C-string of the video is shown in Fig. 4(b).

From frame 7 to 9, video object A changes its motion and moves from left to right along the positive direction of the x-axis with the velocity of 1 unit/frame, but no motion



(a)

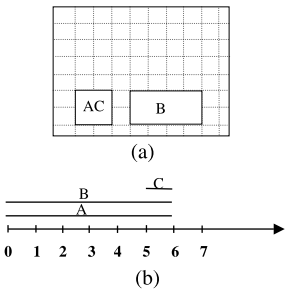
u: $((A_2\uparrow_{0,1} = C_2) <_1 B_4)$

v: $(A_2\uparrow_{1,1} = B_2 = C_2)$

t: $((A_6 = B_6)]\ C_1)$

(b)

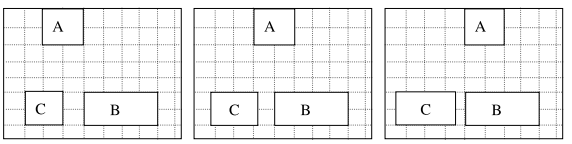
Fig. 2. An example video and the corresponding 3D C-string. (a) A video contains 6 frames; (b) The corresponding 3D C-string.



(a)

(b)

Fig. 3. Projecting the video objects in Fig. 2 onto the x-, y- and time-dimensions: (a) Projecting the initial locations of video objects A, B, and C onto the x–y plane. (b) Projecting the time intervals of video objects A, B, and C onto the time dimension.



(a)

u: $((A_2\uparrow_{0,1}\uparrow_{1,1} = C_2\uparrow_{0,1}\uparrow_{0,1.225}) <_1 B_4)$

v: $(A_2\uparrow_{1,1}\uparrow_{0,1} = B_2 = C_2\uparrow_{0,1}\uparrow_{0,1})$

t: $((A_6|_t A_3 = B_9)]\ C_2|_t C_2)$

(b)

Fig. 4. The video containing three more frames. (a) Three more frames are added to the previous video in Fig. 2; (b) The corresponding 3D C-string.

along the y-axis. Video object C is getting larger along the x-axis with the rate of size change of 1.225 unit/frame, but no size change along the y-axis in frames 7–9. However, there is no motion for the central point of video object C. Because video objects A and C change their states, operator "$|_t$" appears in the t-string of video objects A and C. Therefore, the knowledge structure of 3D C-string provides an easy and efficient way to represent the spatio-temporal relations between the video objects.

Now, we can generate the 3D C-string for the example as shown in Fig. 1. Videos M and N have the same 3D string but their 3D C-strings appear quite different.

Video M:	Video N:
u: $(A_3 \uparrow_{1,1} = B_3)$	u: $(B_6 \%_2 A_2)$
v: $(B_2 <_1 A_2)$	v: $(B_2 <_1 A_2)$
t: $(A_3[B_1)$	t: $(B_3[A_1)$.

By using the 3D C-string representation, the ambiguity problem in 3D string can be resolved. The knowledge structure of 3D C-string provides us more precise descriptions of the motions and size changes of the video objects. The temporal relations between the video objects can be represented in 3D C-string too.

Besides recording the information about the motion of translation and size change of a video object, 3D C-string can be used to represent the motion of rotation of a video object. Let us consider the example shown in Fig. 5(a). In this example, the video contains a still video object (house) and a moving video object (car) with the motion of translation and rotation. Both video objects are approximated by the MBRs.

The corresponding 3D C-string of the video is shown in Fig. 5(b). The central point of video object B moves along the negative direction of the x-axis with the velocity of 92 pixels/frame in frames 1–2. From frame 2 to 3, the central point of video object B moves along the negative direction of the x-axis with the velocity of 59 pixels/frame. The width of video object B is changed from 36 to 38 pixels. So, the rate of size change is $38/36 \cong 1.056$, where "$\cong$" is an approximation operator. From frame 3 to 4, the central point of video object B moves along the negative direction of the x-axis with the velocity of 46 pixels/frame. The width of video object B is changed from 38 to 34 pixels. So, the rate of size change is $34/38 \cong 0.895$. From frame 4 to 5, the central point of video object B moves along the negative direction of the x-axis with the velocity of 30 pixels/frame. The width of video object B is changed from 34 to 22 pixels. So, the rate of size change is $22/34 \cong 0.647$. From frame 5 to 6, the central point of video object B moves along the negative direction of the x-axis with the velocity of 12 pixels/frame. The width of video object B is changed from 22 to 18 pixels. So, the rate of size change is $18/22 \cong 0.818$. Hence, the u-string of the video is $A_{66} <_{36} B_{36} \downarrow_{92,1} \downarrow_{59,1.056} \downarrow_{46,0.895} \downarrow_{30,0.647} \downarrow_{12,0.818}$. Similarly, we can obtain the v-string as shown in Fig. 5(b).

Then, let us consider a video containing a repeated event in which a video object appears above video object C and moves along the positive direction of the y-axis as shown

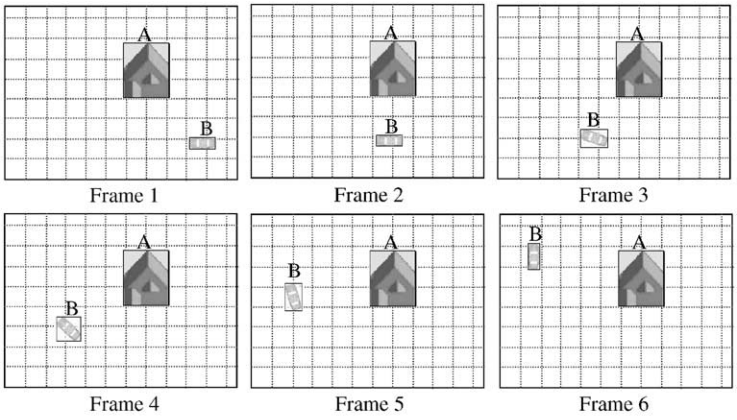


Fig. 5. The video object with the motion of translation and rotation. (a) The video objects are approximated by the MBRs; (b) The corresponding 3D C-string.

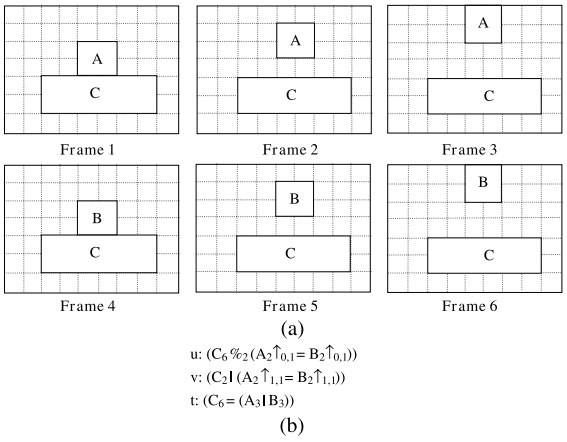


Fig. 6. A video containing a repeated event and its 3D C-string. (a) A video containing a repeated event; (b) The corresponding 3D C-string.

in Fig. 6(a). The corresponding 3D C-string of the video is shown in Fig. 6(b).

From the *u*- and *v*-strings of the video shown in Fig. 6(b), we can see that video objects *A* and *B* have the equal relation and both video objects are followed by the same string of motion and size change. It means that they are of the same size and located at the same spatial location. They also have the same motion and rate of size change. In the *t*-string, the time-projections of video objects *A* and *B* have the "|" relation. It means that video object *B* appears immediately after video object *A* disappears. Hence, a video containing a repeated event can be represented in the knowledge structure of 3D C-string.

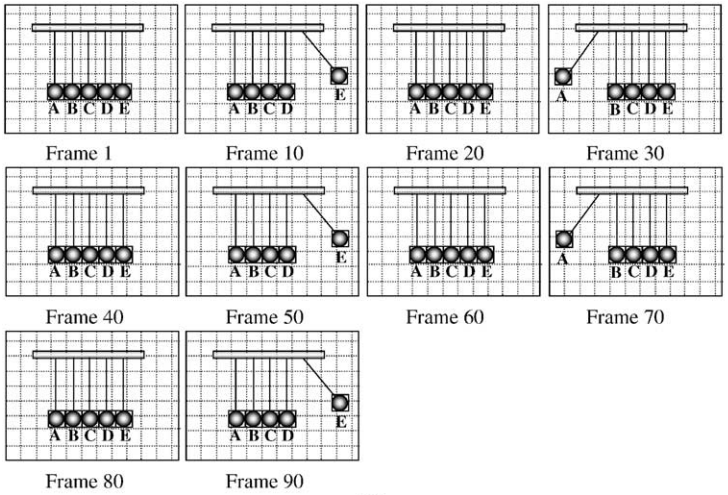
Let us consider another video containing a repeated event in which the video objects (balls) *A* and *E* have cyclic motions as shown in Fig. 7(a). The corresponding 3D C-string of the video is shown in Fig. 7(b).

From the *u*-string of the video, the motion operators following A_{40} form a repeated pattern of $\uparrow_{0,1}\downarrow_{1.2,1}\uparrow_{1.2,1}$. The motion operators following A_{40} in the *v*-string form a repeated pattern of $\uparrow_{0,1}\uparrow_{0.6,1}\downarrow_{0.6,1}$. In the *t*-string, there is a repeated pattern of $A_{20}|_t A_{10}|_t A_{10}$. Similarly, the motion operators following E_{40} in the *u*-string form a repeated pattern of $\uparrow_{1.2,1}\downarrow_{1.2,1}\uparrow_{0,1}$ while that in the *v*-string form a repeated pattern of $\uparrow_{0.6,1}\downarrow_{0.6,1}\uparrow_{0,1}$. In the *t*-string, there is a repeated pattern of $E_{10}|_t E_{10}|_t E_{20}$. Hence, a video containing a cyclic motion can be easily represented in the knowledge structure of 3D C-string.

Therefore, it has been shown that in the knowledge structure of 3D C-string, we can easily manipulate the spatio-temporal relations between the video objects. The knowledge structure of 3D C-string provides us an easy and efficient way to represent the spatio-temporal relations between the video objects in video database systems.

4. String generation algorithm

This section describes a cutting mechanism and string generation algorithm for the knowledge structure of 3D C-string. The string generation algorithm, which is extended



(a)

u: $(A_{40}\uparrow_{0,1}\downarrow_{1.2,1}\uparrow_{1.2,1}\uparrow_{0,1}\downarrow_{1.2,1}\uparrow_{1.2,1}\uparrow_{0,1}|(B_{40}|(C_{40}|(D_{40}|E_{40}\uparrow_{1.2,1}\downarrow_{1.2,1}\uparrow_{0,1}\uparrow_{1.2,1}\downarrow_{1.2,1}\uparrow_{0,1}\uparrow_{1.2,1}))))$

v: $(A_{40}\uparrow_{0,1}\uparrow_{0.6,1}\downarrow_{0.6,1}\uparrow_{0,1}\uparrow_{0.6,1}\downarrow_{0.6,1}\uparrow_{0,1}=B_{40}=C_{40}=D_{40}=E_{40}\uparrow_{0.6,1}\downarrow_{0.6,1}\uparrow_{0,1}\uparrow_{0.6,1}\downarrow_{0.6,1}\uparrow_{0,1}\uparrow_{0.6,1})$

t: $(A_{20}|_tA_{10}|_tA_{10}|_tA_{20}|_tA_{10}|_tA_{10}|_tA_{10}=B_{90}=C_{90}=D_{90}=E_{10}|_tE_{10}|_tE_{20}|_tE_{10}|_tE_{10}|_tE_{20}|_tE_{10})$

(b)

Fig. 7. Another repeated event. (a) The video objects of interest are approximated by the MBRs; (b) The corresponding 3D C-string.

from the concept of the cutting and string generation algorithm proposed by Huang and Jean [12], consists of two parts: spatial string generation and temporal string generation. The basic differences from Huang and Jean's are: (1) in the knowledge structure of 3D C-string, it is required to process temporal relations (t-string) and the information about the motions and size changes of video objects; (2) by introducing the concept of template objects and nearest former objects (described later), the 3D C-string generated by the string generation algorithm is unique.

4.1. Spatial string generation

In the *spatial string generation algorithm*, we introduce a new type of objects: spatial template object. A spatial template object covers the objects enclosed between "(" and ")" separators and is viewed as a new object. The begin-bound of the spatial template object is the smallest begin-bound in all the covered objects. Similarly, the end-bound of the spatial template object is the largest end-bound in all the covered objects. For example, $(A_2 <_3 B_5)$ is a spatial template object whose begin-bound is 0 and end-bound is equal to $2+3+5=10$.

To generate the spatial strings, we first find all the dominating objects, which is defined by Lee and Hsu [9], by scanning the x- (or y-) projections of video objects from left to right along the x- (or y-) axis. The projections of video objects with the same end-bound are grouped into a list. In the list, an object with the smallest begin-bound is called the dominating object. If an object partly overlaps with the dominating object, a cutting is performed at the location of the end-bound of the dominating object. The cutting line is parallel to the x- (or y-) axis. Assume that the projection of a video object O_i (B_i, E_i, L_i) is cut at the location c, where B_i is the begin-bound, E_i is the end-bound, and L_i is a linked list recording the information about the motions and size changes of video object O_i in the x- (or y-) dimension. When the projection of video object O_i is split into two subobjects, the linked list L_i is only given in the leading subobject. So, the projection of video object O_i is split into two subobjects O_i (B_i, c, L_i) and O_i (c, E_i).

By scanning from left to right along the x- (or y-) axis, we shall find all dominating objects. For each dominating object, the objects covered by the dominating object are merged into a spatial template object. Finally, we can merge together those spatial template objects and the remaining objects not covered by any dominating objects. How to merge objects into a spatial template object is described later in the *spatial template object generation algorithm*. This is the main idea of the *spatial string generation algorithm*.

The *spatial string generation algorithm* is described in detail as follows:

Algorithm: Spatial string generation

Input: O_1 (B_1, E_1, L_1), O_2 (B_2, E_2, L_2), O_3 (B_3, E_3, L_3), …, O_n (B_n, E_n, L_n)

Output: An u-string (or v-string)

1. Sort in non-decreasing order all the begin-bound and end-bound points B_i, E_i, $i = 1, 2, \ldots, n$.
2. Group the same value points into a same-value list. Form a same-value-list sequence.
3. Loop from step 4 to step 8 for each same-value list according to non-decreasing order.
4. If there is no end-bound in the list, process the next same-value list.
5. Find the dominating object from the objects in the same end-bound list so that the begin-bound of the dominating object is the smallest of them. If an object partly overlaps with the dominating object, a cutting is performed at the location of the end-bound of the dominating object. The cutting line is parallel to the x- (y-) axis.
6. Compute the size of the dominating object. If the linked list, L, of the dominating object is not *null*, call the *msc-string generation algorithm* and merge the generated msc-string to the representation of the dominating object. If there exist objects partly overlapping with the dominating object, perform the following two phases.

   (a) The latter objects partly overlapping with the dominating object are segmented. The size of the first subobject of a segmented object is equal to the end-bound of the dominating object subtracted by the begin-bound of the segmented object.
   (b) For each segmented object, if its linked list is not *null*, call the *msc-string generation algorithm* and merge the generated msc-string to the representation of the leading subobject of the segmented object. The remaining subobjects of segmented objects and objects whose begin-bounds are at the cutting line are viewed as the new objects with the begin-bound at the location of the cutting line. Mark them with an "edge" flag.

7. Find the list of objects covered by the dominating object and call the *spatial template object generation algorithm* with the found object list as the input parameter.
8. Collect the begin-bound and the end-bound of the new template object into the same-value lists.
9. Call the *spatial template object generation algorithm* with the object list formed by all remaining objects as the input parameter. Output the representation of the final object. This is the u- (or v-) string.

Before introducing the *spatial template object generation algorithm*, we define some terminology used in the algorithm. A former object of object O is an object with smaller begin-bound than that of object O, or an object with equal begin-bound and bigger end-bound than that of object O. The nearest former object is the former object with the biggest begin-bound. If the number of such objects is more than 1, choose one with the smallest end-bound as the nearest former object. Hence, for each object, its nearest former object is unique.

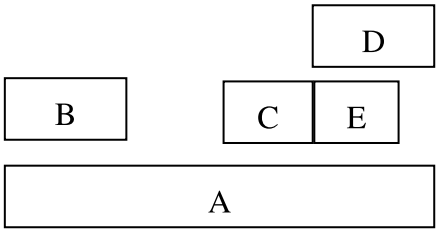


Fig. 8. Object E is the object that is not any objects' former objects.

Given a list of objects, there exists an object, Q, that is not any objects' former objects. The begin-bound of object Q should be the largest among those of the objects in the list. If the number of objects with the largest begin-bound is more than 1, object Q should be the object with the smallest end-bound. If the number of such objects is more than 1, it means that they have the same begin-bound and end-bound and they can be merged together by operator "=" into a template object. The template object is viewed as a new object. Let Q be the merged template object. Hence, for a list of objects, there exists unique object, Q, that is not any objects' former objects.

Now, let us consider the example as shown in Fig. 8. Object A is the nearest former object of object B. Object B is the nearest former object of object C. Object C is the nearest former object of object D. Object D is the nearest former object of object E. Object E is the object that is not any objects' former objects. The begin-bound of the x-projections of objects D and E is the largest among objects A, B, C, D and E. However, the end-bound of the x-projection of object E is smaller than that of object D.

Actually, we can use the relationship of the nearest former object to decide the order of merging objects into a spatial template object. In the example as shown in Fig. 8, objects D and E are first merged into a template object O_1 since object E is not any objects' former objects and D is the nearest former object of E. Second, objects O_1 and C are merged into a template object O_2 since object O_1 is not any objects' former objects and object C is the nearest former object of object O_1. Similarly, objects O_2 and B are merged into a template object O_3 since object O_2 is not any objects' former objects and object B is the nearest former object of object O_2. Finally, objects O_3 and A are merged into a template object O_4. The corresponding u-string of object O_4 is $(A = (B < (C|(D[E))))$ where the metric information is omitted. How to merge objects into a spatial template object is described in detail as follows.

Algorithm: Spatial template object generation

Input: A list of objects
Output: A spatial template object

1. Repeat steps 2–5 until there is only one object in the list.
2. For the objects having the same begin-bound and end-bound, they are chained by "=" operator and form a spatial template object. If there is only one object in the list, exit the repeat-loop.
3. For each object, find its nearest former objects.
4. Let Q be the object that is not any objects' former objects and N be the nearest former object of object Q. Perform the following phases to find an appropriate operator to merge object Q with object N.

   (a) If object N satisfies the following two conditions: (1) its begin-bound is the same as that of object Q and (2) its end-bound is bigger than that of object Q, use "[" operator to merge objects Q and N. Go to step 5.
   (b) If object N satisfies the following two conditions: (1) its end-bound is the same as that of object Q and (2) its begin-bound is smaller than that of object Q, use "]" operator to merge objects Q and N. Go to step 5.
   (c) If the end-bound of object N is smaller than the begin-bound of object Q, use "$<$" operator to merge objects Q and N. The distance associated with the "$<$" operator is equal to the begin-bound of object Q subtracted by the end-bound of object N. Go to step 5.
   (d) If the end-bound of object N is equal to the begin-bound of object Q, use "|" operator to merge objects Q and N. Go to step 5.
   (e) If object N has bigger end-bound than that of object Q, use "%" operator to merge objects Q and N. The distance associated with this "%" operator is equal to the begin-bound of object Q subtracted by the begin-bound of object N.

5. If either object Q or object N is not a spatial template object, compute the size of the object. If the linked list, L, of the object is not *null*, call the *msc-string generation algorithm* and merge the generated msc-string to the representation of the object. Then objects Q and N are merged into a new spatial template object by "(" and ")" separators with the appropriate operator found in step 4. The begin-bound of the spatial template object is the smaller begin-bound of objects Q and N. The end-bound of the spatial template object is the larger end-bound of objects Q and N.

Algorithm: msc-string generation

Input: $O(B, E, L)$
Output: A motion and size change string (msc-string for short)

1. If L is not *null*, generate "↑" or "↓" operator to represent the motion and size change for each node of L depending on the positive or negative moving direction, velocity and rate of size change associated with O.

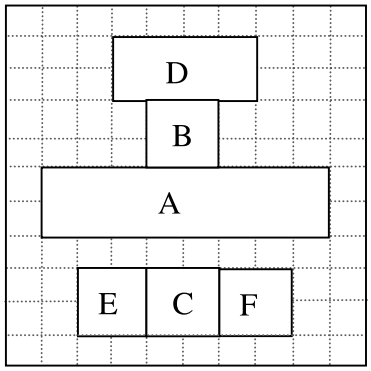


Fig. 9. The initial locations of video objects A, B, C, D, E and F are projected onto the x and y dimensions.

2. Merge those generated "↑" or "↓" operators into an msc-string.

Now, let us consider the example as shown in Fig. 9. For simplicity, we do not consider the motions in the example. The initial locations of the six video objects in the video can be represented with their begin-bound and end-bound points in the x dimension and form an object list as follows:

$$A(1,9), B(4,6), C(4,6), D(3,7), E(2,4), F(6,8).$$

Then we demonstrate how we apply the *spatial string generation algorithm* to the above object list in order to obtain an u-string.

First, a cutting is performed at the location of the end-bound of the dominating object E in step 5 of the *spatial string generation algorithm*. Object D is segmented into two subobjects. In step 7, we find the list of objects covered by the dominating object E and call the *spatial template object generation algorithm* with the found object list as the input parameter. In this case, the object list contains object E and the first subobject of D. The *spatial template object generation algorithm* then outputs a template object. Its representation is $(E_2]D_1)$.

Second, a cutting is performed at the location of the end-bound of the dominating object D in step 5 of the *spatial string generation algorithm*. Object F is segmented into two subobjects. In step 7, we find the list of objects covered by the dominating object D (the second subobject of D) and call the *spatial template object generation algorithm* with the found object list as the input parameter. In this case, the object list contains objects B and C, the second subobject of D and the first subobject of F. In the first repeat-loop of the *spatial template object generation algorithm*, steps 2–3 and step 4(d) are executed. It generates a spatial template object with the representation of $((B_2 = C_2)|F_1)$. This is because subobject F is the only object that is not any objects' former objects in the list and spatial template object $(B_2 = C_2)$

is F's nearest former object. In the second repeat-loop of the *spatial template object generation algorithm*, step 2 is executed. It generates the spatial template object with the representation of $(D_3 = ((B_2 = C_2)|F_1))$. This is because the size of the second subobject of D is equal to that of the spatial template object generated by the first repeat-loop. Then output this spatial template object.

Third, at this point, because no further cuttings are needed, step 9 of *the spatial string generation algorithm* is executed. In this case, we call the *spatial template object generation algorithm* with an object list formed by all remaining objects as the input parameter.

The second subobject of F is the only object that is not any objects' former objects. So it is merged with its nearest former object as a new spatial template object in the first repeat-loop. The representation of the spatial template object is $((D_3=((B_2=C_2)|F_1))|F_1)$. In the second repeat-loop, two spatial template objects $(E_2]D_1)$ and $((D_3 = ((B_2 = C_2)|F_1))|F_1)$ are merged together as a new spatial template object in step 4(d). Its representation is $((E_2]D_1)|((D_3 = ((B_2 = C_2)|F_1))|F_1))$. In the fourth repeat-loop, two objects A_8 and $((E_2]D_1)|((D_3 = ((B_2 = C_2)|F_1))|F_1))$ are merged together as a new spatial template object in step 4(e). Its representation is $(A_8\%_1((E_2]D_1)|((D_3 = ((B_2 = C_2)|F_1))|F_1)))$.

Finally, all objects are merged together as a spatial template object. So, the corresponding u-string of the video as shown in Fig. 9 can be represented as $(A_8\%_1((E_2]D_1)|((D_3= ((B_2 = C_2)|F_1))|F_1)))$.

Lemma 1. *For an input list containing n objects, the representation, u- (or v-) string, of the spatial template object generated by the* spatial template object generation algorithm *is unique.*

Proof. We prove the lemma by mathematical induction on the number of objects in the input list.

Basis step: The lemma is trivially true for $n = 1$. The representation, u- (or v-) string, of the object is the object symbol associated with its size and the information of its motions and size changes if the object changes its states in the video.

Induction hypothesis: The lemma is true for all videos containing j objects and $j \leqslant k$. That is, the representation, u- (or v-) string, of the template object generated by the *spatial template object generation algorithm* is unique for the input list containing j objects, $j \leqslant k$.

Induction step: Consider the input list containing $k + 1$ objects, and there exists a unique object O that is not any objects' former objects. For object O, there exists a unique object N that is the nearest former object of object O. The way of merging object O and object N is performed by one of the following steps.

(a) If object N satisfies the following two conditions: (1) its begin-bound is the same as that of object O and (2) its end-bound is bigger than that of object O, use "[" operator to merge objects O and N.
(b) If object N satisfies the following two conditions: (1) its end-bound is the same as that of object O and (2) its begin-bound is smaller than that of object O, use "]" operator to merge objects O and N.
(c) If the end-bound of object N is smaller than the begin-bound of object O, use "$<$" operator to merge objects O and N. The distance associated with the "$<$" operator is equal to the begin-bound of object O subtracted by the end-bound of object N.
(d) If the end-bound of object N is equal to the begin-bound of object O, use "|" operator to merge objects O and N.
(e) If object N has bigger end-bound than that of object O, use "%" operator to merge objects O and N. The distance associated with this "%" operator is equal to the begin-bound of object O subtracted by the begin-bound of object N.

For either one of objects O and N, if its linked list is not *null*, call the *msc-string generation algorithm* and merge the generated msc-string to the representation of the object. Then objects O and N are merge into a spatial template object by "(" and ")" separators. The way of merging objects O and N is performed by one of the above steps. So the representation of the spatial template object is unique. Because objects O and N are merged into one spatial template object, there are k objects to be processed. The string generated to represent the remaining k objects is unique by the induction hypothesis. So we can prove that for an input list containing n objects, the representation, u- (or v-) string, of the spatial template object generated by the *spatial template object generation algorithm* is unique. □

Lemma 2. *For an input list containing n objects, the* spatial string generation algorithm *generates a unique u- (or v-) string.*

Proof. For each dominating object, in step 5, if an object partly overlaps with the dominating object, a cutting is performed at the location of the end-bound of the dominating object. The cutting line is parallel to the x- (y-) axis. In step 7, we find all the objects covered by the dominating object and call the *spatial template object generation algorithm* with the found object list as the input parameter. By Lemma 1, we know that the representation, u- (or v-) string, of the spatial template object generated by the *spatial template object generation algorithm* is unique.

By scanning from left to right along the x- (or y-) axis, we shall find all dominating objects. For each dominating object, the objects covered by the dominating object are merged into a spatial template object. The representation of the spatial template object is unique. In step 9, we call the *spatial template object generation algorithm* with the object list formed by those spatial template objects and the remaining

objects not covered by any dominating objects as the input parameter. By Lemma 1, we know that the representation, u- (or v-) string, of the spatial template object generated by the *spatial template object generation algorithm* is unique, too. Finally, the *spatial string generation algorithm* outputs the representation of the final object. Therefore, the *spatial string generation algorithm* generates a unique u- (or v-) string for an input list containing n objects. □

4.2. Temporal string generation

The *temporal string generation algorithm* is similar to the *spatial string generation algorithm*. The major difference between both algorithms is that the *temporal string generation algorithm* needs to process the partitioning points and does not need to process the information of motions and size changes of objects.

Assume that the time-projection of object O_i $(B_i, T_i^1, T_i^2, T_i^3, \ldots, T_i^k, E_i)$, $k \geqslant 1$, is partitioned at time points $T_i^1, T_i^2, T_i^3, \ldots, T_i^k$, where B_i and E_i are the begin-bound and end-bound of the time-projection of object O_i. T_i^j, $1 \leqslant j \leqslant k$, $k \geqslant 1$ is the jth time point of changing the state of object O_i. If the time-projection of object O_i is equal to (B_i, E_i), it means that there are no partitions in the time interval between B_i and E_i. That is, there are no motions or size changes during the time interval between B_i and E_i. Hence, if the time-projection of object O_i is equal to $(B_i, T_i^1, T_i^2, T_i^3, \ldots, T_i^k, E_i)$, $k \geqslant 1$, it means that the time interval between B_i and E_i is partitioned into $k+1$ parts. There exists a certain motion or size change in each part.

We shall generate a "$|_t$" operator for each time point when the state of an object is changed. If a cutting is performed, we split the object into two subobjects at the cutting point. For example, if a cutting is performed at point c which is between B_i and T_i^1, O_i (B_i, T_i^1) is split into O_i (B_i, c) and O_i (c, T_i^1).

Since the *temporal string generation algorithm* is quite similar to the *spatial string generation algorithm*, it is omitted.

Lemma 3. *For an input list containing n objects, the* temporal string generation algorithm *generates a unique t-string.*

Proof. The proof is similar to that of Lemma 2. □

Theorem 1. *For an input list containing n objects, the* string generation algorithm *generates a unique* 3*D C-string.*

Proof. By Lemma 2, we know that the u- and v-strings generated by the *spatial string generation algorithm* are unique. By Lemma 3, we know that the t-string generated by the *temporal string generation algorithm* is unique, too. So, the 3D C-string is unique. Therefore, for an input list containing n objects, the *string generation algorithm* generates a unique 3D C-string. □

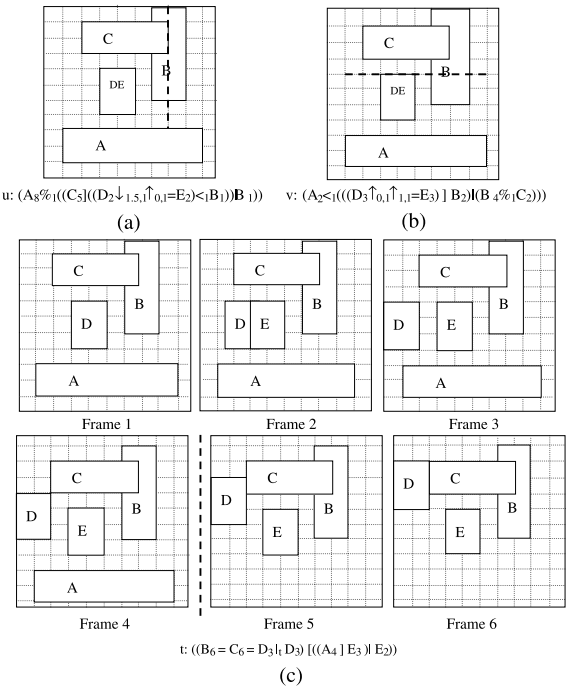


Fig. 10. Example of cutting and string generation. (a) Cutting along the x-axis; (b) Cutting along the y-axis; (c) Cutting along the time-axis.

Now, let us consider the example as shown in Fig. 10. The cutting and the u-string generated by the *spatial string generation algorithm* are shown in Fig. 10(a) where the dot line is the cutting line. All initial locations and sizes of the objects in the video are projected onto the x-axis and form a frame as shown in Fig. 10(a). Similarly, the cutting along the y-axis and the corresponding v-string are shown in Fig. 10(b). The cutting and the t-string generated by the *temporal string generation algorithm* are shown in Fig. 10(c). We can see that the cutting is performed between frames 4 and 5.

5. Video reconstruction algorithm

This section presents a video reconstruction algorithm which converts a 3D C-string into a symbolic video for visualization and browsing of video databases. The video reconstruction algorithm consists of two parts: spatial string reconstruction and temporal string reconstruction. The *spatial string reconstruction algorithm* processes an u-string (or v-string) to construct the locations, motions, and size changes for video objects in a symbolic video. The *temporal string reconstruction algorithm* processes

the corresponding t-string to construct the duration of motions and size changes of video objects in the time dimension.

5.1. Spatial string reconstruction

In the *spatial string reconstruction algorithm*, we introduce the notations of a spatial string object S and a video object O in our algorithm. Suppose that a u-string (or v-string) consists of n elements, each of which may be a spatial string object, a relation operator, or a motion operator. If element E is a spatial string object, E.sym represents the string symbol and E.size represents the size associated with E. The operators "$<$", "%", "↑", and "↓" also have metric information associated with them. If element E is one of the relation operators "%" or "$<$", E.sym represents the operator symbol and E.size represents the distance associated with them. If element E is a motion operator ("↑" or "↓"), it has two fields, (v, r), to record the velocity and rate of size change for the associated spatial string object. For the operators other than "$<$", "%", "↑", and "↓", the size fields associated with them are set to zero. Similarly, a video object O contains four fields: O.sym, O.size, O.location, and O.motionList. O.sym, O.size, and O.location represent the symbol, size, and the location of object O in its starting frame, respectively. An element in O.motionList consists of two fields, (v, r), to record the velocity and rate of size change of object O.

Assume that there are n elements in a given u-string (or v-string) and m spatial string objects in the n elements. The *spatial string reconstruction algorithm* converts the given u-string (or v-string) into a list of m video objects. After both lists of video objects are derived from the given u- and v-strings, we have finished the spatial part of the video reconstruction. The *spatial string reconstruction algorithm* is described in detail as follows:

Algorithm: Spatial string reconstruction
Input: A u-string (or v-string) with n elements: string $= (E_1, E_2, \ldots, E_n)$
Output: A list of video objects: ObjectList$= (O_1, O_2, \ldots, O_m)$

```
/* Initialization */
1. Loc ← 0; ObjectList ←nil; Stack ←nil; i ← 1; j ← 0;
2. MoreOperators ← False;
3. while (more elements in the u- (or v-) string)
       /* process the u- or v-strings */
4.     while (MoreOperators)
5.     i ← i + 1;    /* next operator */
6.     case E_i.sym
7.       "%"□Loc ← Loc + E_i.size;
8.             i ← i + 1;
9.             MoreOperators ← False;
10.      "<"□Loc ← Loc + PreviousObjectSize +
                          E_i.size;
11.            i ← i + 1;
12.            MoreOperators ← False;
13.      "|"□Loc ← Loc + PreviousObjectSize;
14.            i ← i + 1;
15.            MoreOperators ← False;
16.      "]"□If E_{i+1}.sym ≠"(" then TemplateSize ←
                   E_{i+1}.size;
17.            else TemplateSize ←
                   GetTemplateSize (i + 1, string);
18.            end-if
19.            Loc ← Loc+PreviousObjectSize
                   −TemplateSize;
20.            i ← i + 1;
21.            MoreOperators ← False;
22.      "=" or "[": i ← i + 1;
23.            MoreOperators ← False;
24.      "↑"□Append (v, r) to O_j.motionList;
25.            MoreOperators ← True;
26.      "↓"□Append (−v, r) to O_j.motionList;
27.            MoreOperators ← True;
28.      ")"□Pop an element E from Stack;
             /* E is a template object */
29.            Loc ← E.beginBound;
30.            PreviousObjectSize ← E.size;
31.            MoreOperators ← True;
32.    end-case
33.  end-while
34.  while (E_i.sym = "(")
35.    Create a template object E;
36.    E.beginBound ← Loc;
37.    E.size ← GetTemplateSize(i,string);
38.    Push the template object E into Stack;
39.    i ← i + 1;
40.  end-while     /* Index to the next string object */
41.  if E_i is the leading subobject of an object then
42.    j ← j + 1;
43.    Create a new object O_j so that
44.          O_j.sym ← E_i.sym;
45.          O_j.size ← E_i.size;
46.          O_j.beginBound ← Loc;
47.    Append object O_j to ObjectList.
48.  else O_j.size ← O_j.size + E_i.size;
       /* Update object's size */
49.  end-if
50.  PreviousObjectSize ← E_i.size;
51.  MoreOperators ← True;
52. end-while
53. Output the ObjectList.
```

The function GetTemplateSize calculates the size of the template object at the next level which is the summation of:

1. The size of the first element after "(".
2. The size of the element after global operator "$<$".

3. The size of the element after global operator "|".
4. The distance associated with the operator "<".

Notice that it is not necessary to calculate the sizes of template objects at third or lower levels.

Now, we can prove that for a given u-string (v-string), the list of video objects generated by the *spatial string reconstruction algorithm is unique.*

Lemma 4. *For a given u-string* (*or v-string*), *the list of video objects generated by the* spatial string reconstruction algorithm *is unique*.

Proof. We prove the lemma by mathematical induction on the number of levels of spatial template objects, p.

Basis step: The lemma is trivially true for $p = 0$. The reconstruction of a string with zero level is to construct a video object for a spatial string object with its associated size, starting location, and motion list.

Induction hypothesis: The lemma is true for all strings having p levels of template objects and $p \leqslant k$, that is, the spatial string reconstruction algorithm generates a unique list of video objects for the input string of p levels of spatial template objects, $p \leqslant k$.

Induction step: Consider an input string having $k + 1$ levels of spatial template objects, $k \geqslant 0$. Assume that there are q objects between the first "(" symbol and the last ")" symbol, $q > 0$. Some of them are string objects and the others are spatial template objects. The number of levels of those spatial template objects should be $\leqslant k$. The spatial string reconstruction algorithm processes those string objects and spatial template objects from left to right.

Let us number those objects, including string objects and spatial template objects, from 1 to q. Hence, the algorithm processes those objects one by one from the first object to the qth object. If the ith object is a spatial string object, the algorithm will create a new video object for it and set the begin-bound, size, motions and size changes for the newly created video object. If the ith object is a spatial template object, the number of levels of the spatial template object is $\leqslant k$. By the induction hypothesis, the algorithm will generate a unique list of video objects for the spatial template object. Depending on the operator following the ith object, the algorithm can decide the starting location (the variable "Loc" in the algorithm) of the $(i + 1)$th object. There are five cases.

(a) If the following operator is the "%" symbol, Loc←Loc $+\mathrm{E}_i$.size. That is, the starting location of $(i+1)$th object is equal to the starting location of the ith object plus the distance associated with the "%" symbol.
(b) If the following operator is the "<" symbol, Loc←Loc + PreviousObjectSize + E_i.size. That is, the starting location of the $(i + 1)$th object is equal to the starting location of the ith object plus the size of the ith object plus the distance associated with the "<" symbol.
(c) If the following operator is the "|" symbol, Loc←Loc+ PreviousObjectSize. That is, the starting location of the $(i + 1)$th object is equal to the starting location of the ith object plus the size of the ith object.
(d) If the following operator is the "]" symbol, Loc←Loc + PreviousObjectSize − TemplateSize. That is, the starting location of the $(i+1)$th object is equal to the starting location of the ith object plus the size of the ith object minus the size of $(i + 1)$th object.
(e) If the following operator is the "=" or "[" symbols, the starting location of the $(i + 1)$th object is the same as that of the ith object.

Repeat the above procedure from $i = 1$ to q. So, the algorithm can generate a unique list of video objects for each object (either a string object or a spatial template object). Therefore, we can prove that for a given u-string (or v-string), the list of video objects generated by the spatial string reconstruction algorithm is unique. □

5.2. Temporal string reconstruction

The *temporal string reconstruction algorithm* is similar to the *spatial string reconstruction algorithm.* The major difference of both algorithm is that the temporal string reconstruction algorithm needs to process "$|_t$" operator but it does not need to process "↑" and "↓" operators. Hence, the temporal string reconstruction algorithm is omitted.

Lemma 5. *For a given t-string*, *the list of video objects generated by the* temporal string reconstruction algorithm *is unique*.

Proof. The proof is similar to that of Lemma 4. □

Theorem 2. *For a given 3D C-string*, *the lists of video objects generated by the video reconstruction algorithm are unique*.

Proof. From Lemma 4, we know that for the given u- and v-strings, the lists of video objects generated by the *spatial string reconstruction algorithm* are unique. From Lemma 5, we know that for the given t-string, the list of video objects generated by the *temporal string reconstruction algorithm* is unique too. Therefore, for a given 3D C-string, the lists of video objects generated by the video reconstruction algorithm are unique. □

After finishing the spatial and temporal parts of video reconstruction algorithm, we can draw a symbolic video very easily based on the starting location, size, starting frame number, duration, information about the motions and rates of size changes, and the duration of states of each video object.

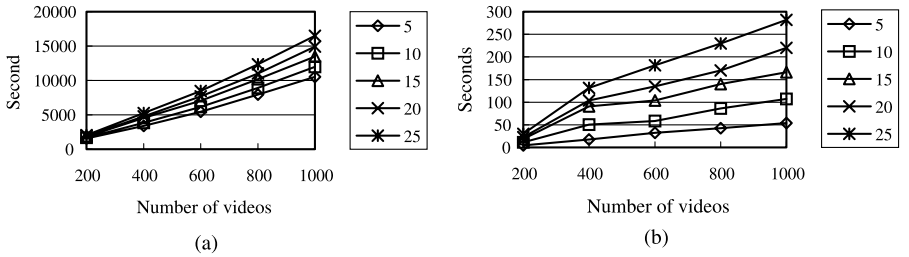


Fig. 11. The execution time vs. the number of videos: (a) string generation and (b) video reconstruction.

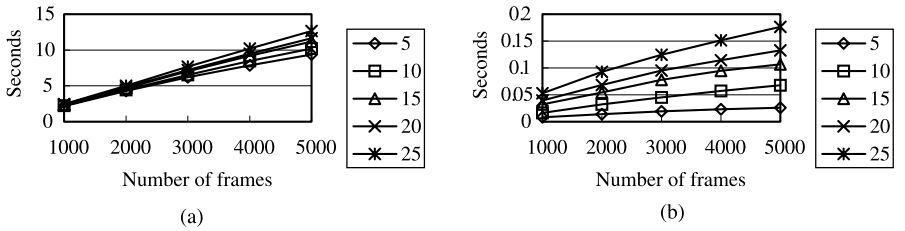


Fig. 12. The execution time vs. the number of frames in a video: (a) string generation and (b) video reconstruction.

6. Performance analysis

To show the performance of our *string generation and video reconstruction algorithms*, we perform two sets of experiments. The first set of experiments is made on the synthesized videos. There are three cost factors dominating the performance of the *string generation and video reconstruction algorithms*: the number of videos, the number of video objects, the number of frames in a video. We freely set the values of the three cost factors in the synthesized videos. The second set of experiments is made on 200 real videos. Each video is clipping of about 1 min. The video objects in each video are specified by using the video index tool. All the algorithms are implemented on an IBM compatible personal computer of Pentium III-800 with Windows 2000.

6.1. Synthesized videos

In this subsection, we show the performance of our *string generation and video reconstruction algorithms*. The execution cost of every experiment is measured by the elapsed time of video processing. We generate the video indices for 5000 videos. For each video, we assign 25 objects and 5000 frames to it. Based on these synthesized videos, we perform four experiments. The experimental results are shown as follows.

Fig. 11 illustrates the execution time versus the number of videos for the *string generation and video reconstruction algorithms*. Each video in these two experiments contains 4000 frames. The execution time grows as the number of videos increases.

Table 2
The average number of video objects for each type of videos

Type	Campus	Cartoon	Traffic	TV news
Average number of video objects	18	28	64	31

Fig. 12 illustrates the execution time versus the number of frames in a video for the string generation and video reconstruction algorithms. In these two experiments, we run 200 videos for each case. The execution time is averaged over the execution time for each video. The execution time grows nearly linear as the number of frames in a video increases.

6.2. Real videos

In this subsection, we show the performance of our *string generation and video reconstruction algorithms* with real videos. Since the performance of both algorithms depends on the number of video objects in a real video, the average number of video objects contained in a real video is 37. In our example video database, there are video objects of cars, people, and buildings. The example video database contains four types of videos: 60 videos of traffic, 60 videos of campus activities, 40 videos of cartoons and 40 videos of TV news. There are 200 videos in total. All videos are around 1 min long. In general, we specify 1–10 video objects from each frame. Typically, a video of 1 min long contains 1800 frames. To represent the movements of video objects, at least a frame should be indexed for every 10 frames. The average number of video objects for each type of videos is shown in Table 2.

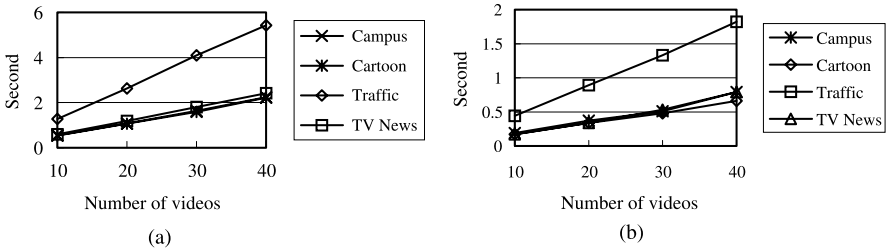


Fig. 13. The execution time vs. the number of videos: (a) string generation and (b) video reconstruction.

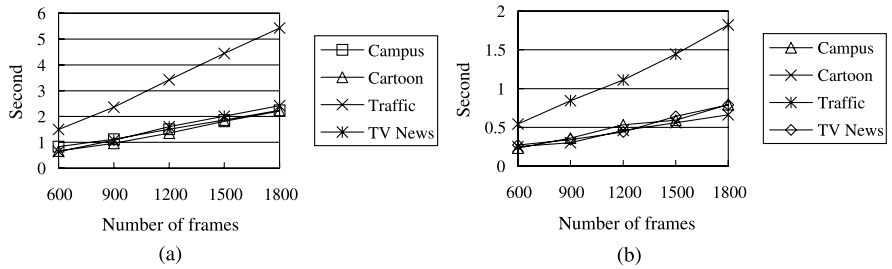


Fig. 14. The execution time vs. the number of frames: (a) string generation and (b) video reconstruction.

Fig. 13 illustrates the execution time versus the number of videos for the *string generation and video reconstruction algorithms* for each type of videos. The execution time grows linear as the number of videos increases. The execution time of processing traffic videos is biggest among them since such a type of videos contains the most number of video objects. The execution time of the other three types of videos is quite close. Although the videos of campus activities contain the least number of video objects, the motion and size change operators in such a type of videos are longest. Hence, it takes time to generate a 3D C-string and to reconstruct a video.

Fig. 14 illustrates the execution time versus the number of frames for the *string generation and video reconstruction algorithms* for each type of videos. The execution time grows as the number of frames increases. In both figures, we use 40 videos for each type of videos.

It is shown that the different types of videos have different results for the cost of generating and reconstructing a video in the real video database. For the videos of traffic, they contain the most number of video objects. So, the execution time of this type is largest among them. Since the average number of video objects of cartoons is close to that of TV news, the execution time of both types is quite close too. The average number of video objects of campus activities is the smallest among them. However, a video object in this type of video changes its states quite often including motions or rates of size changes. Hence, the 3D C-strings generated by this type of videos contain a lot of motion operators. The execution time of the video of campus activities is higher than expected.

7. Concluding remarks

In this paper, we propose a new spatio-temporal knowledge representation called 3D C-string for video database systems. Since 3D string is not powerful enough to describe the spatial knowledge of non-zero sized objects and temporal relations between the objects in a video, it is not suitable for processing videos with overlapping objects and temporal events. It cannot represent the spatio-temporal knowledge between the objects precisely. 3D C-string extends the concepts of 2D C^+-string to overcome the weakness of 3D string. We propose the string generation algorithm to generate a 3D C-string for a given video and the video reconstruction algorithm to reconstruct a symbolic video from a given 3D C-string. By introducing the concept of the template objects and nearest former objects, the string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D C-string is unique too. The 3D C-string representation captures not only the spatial relations but temporal relations between the objects in a video. Our new representation method can be easily applied to an intelligent video database system to reason about spatio-temporal relations between the objects in a video.

8. Summary

In video database systems, one of the most important methods for discriminating the videos is by using the objects and the perception of spatial and temporal relationships that exist among objects in the desired videos. Therefore, how

videos are stored in a database becomes an important design issue of video database systems. The spatio-temporal knowledge embedded in videos should be preserved in the data structure and the data structure should be object-oriented, so that users can easily retrieve, visualize and manipulate objects in the video database systems. In comparison with text, image, and audio, video contains richer information. But the richness results in the lack of generally accepted representation of a video.

The knowledge structure called 2D C^+-string to represent symbolic images was proposed by P.W. Huang et al. It allows us to represent spatial knowledge in images. The knowledge structure called 3D string to represent the spatial and temporal relationships among symbolic video objects was proposed by Liu and Chen. In 3D string representation, an object is represented by its central point and the starting frame number of the object. So they cannot realize the spatial overlapping relations and precise temporal relations. The information about the motions and sizes of objects is omitted in this work.

Therefore, we need one more compact and precise knowledge structure to represent spatio-temporal relationships among objects and to keep track of the motions and size changes associated with the objects in a video.

In this paper, we propose a new spatio-temporal knowledge representation called 3D C-string. The knowledge structure of 3D C-string, extended from the 2D C^+-string, uses the projections of objects to represent spatial and temporal relations between the objects in a video. Moreover, it can keep track of the motions and size changes of the objects in a video. The string generation and video reconstruction algorithms for the 3D C-string representation of video objects are also developed. By introducing the concept of the template objects and nearest former objects, the string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D C-string is unique too. This approach can provide us an easy and efficient way to retrieve, visualize and manipulate video objects in video database systems. Finally, some experiments are performed to show the efficiency of the proposed algorithms.

References

[1] S. Gibbs, C. Breitender, D. Tsichritzis, Audio/video database: an object-oriented approach, in: Proceedings of the IEEE International Conference on Data Engineering, 1993, pp. 381–390.

[2] Y. Tonomura, et al., Structured video computing, IEEE Multimedia 1 (3) (1994) 34–43.

[3] E. Oomoto, K. Tanaka, OVID: design and implementation of a video-object database system, IEEE Trans. Knowledge Data Eng. 5 (4) (1993) 629–643.

[4] R. Weiss, A. Duda, D.K. Gifford, Content-based access to algebraic video, in: Proceedings of the ACM International Conference on Multimedia Computing and Systems, May 1994, pp. 140–151.

[5] S.W. Smoliar, H.J. Zhang, Content-based video indexing and retrieval, IEEE Multimedia 1 (2) (1994) 62–72.

[6] S.K. Chang, Q.Y. Shi, C.W. Yan, Iconic indexing by 2D strings, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-9 (1987) 413–429.

[7] S.K. Chang, E. Jungert, A spatial knowledge structure for image information systems using symbolic projections, in: Proceedings, Fall Joint Computer Conference, Dallas, TX, November 1986, pp. 79–86.

[8] S.K. Chang, E. Jungert, Y. Li, Representation and retrieval of symbolic pictures using generalized 2D strings, Technical Report, University of Pittsburgh, 1988.

[9] S.Y. Lee, F.J. Hsu, 2D C-string: a new spatial knowledge representation for image database system, Pattern Recognition 23 (1990) 1077–1087.

[10] S.Y. Lee, F.J. Hsu, Picture algebra for spatial reasoning of iconic images represented in 2D C-string, Pattern Recognition Lett. 12 (1991) 425–435.

[11] S.Y. Lee, F.J. Hsu, Spatial reasoning and similarity retrieval of images using 2D C-string knowledge representation, Pattern Recognition 25 (1992) 305–318.

[12] P.W. Huang, Y.R. Jean, Using 2D C^+-string as spatial knowledge representation for image database systems, Pattern Recognition 27 (1994) 1249–1257.

[13] P.W. Huang, Y.R. Jean, Spatial reasoning and similarity retrieval for image database systems based on RS-strings, Pattern Recognition 29 (1996) 2103–2114.

[14] C.C. Liu, A.L.P. Chen, 3D-list: a data structure for efficient video query processing, IEEE Trans. Knowledge Data Eng., to appear.

[15] E. Jungert, Extended symbolic projections as a knowledge structure for spatial reasoning, in: Proceedings of the Fourth BPRA Conference on Pattern Recognition, March 1988, pp. 343–351.

[16] E. Jungert, S.K. Chang, An algebra for symbolic image manipulation and transformation, in: T.L. Kunii (Ed.), Visual Database Systems, Elsevier Science Publishers B.V. (North-Holland), IFIP, Amsterdam, 1989.

About the Author—ANTHONY J.T. LEE was born on 26 June 1961 in Taiwan R.O.C. He received the B.S. degree from National Taiwan University, Taiwan, in 1983. He got the M.S. and Ph.D. degrees in Computer Science from University of Illinois at Urbana-Champaign, USA, in 1990 and 1993, respectively. In August 1993, he joined the Department of Information Management at National Taiwan University and is now an associate professor. His current research interests include multimedia databases, temporal and spatial databases, and data mining. Dr. Lee is a member of the IEEE Computer Society.

About the Author—HAN-PANG CHIU was born on 7 November 1978 in Taipei, Taiwan, R.O.C. He received the B.B.A degree from National Taiwan University, Taiwan, in 1999 and the MBA degree from Department of Information Management, National Taiwan University

in June 2001. He is going to join the Ph.D. program of the Department of Computer Science, Massachusetts Institute of Technology, USA, in September 2001.

About the Author—PING YU was born on 30 June 1966 in Taipei, Taiwan, R.O.C. He received the B.S degree from Chung Cheng Institute of Technology, Taiwan, in 1988, and the M.S. degree from National Defense Management College, Taiwan, in 1994. He is currently working for his Ph.D.'s degree in the Department of Information Management, National Taiwan University.