3D-List: A Data Structure for Efficient Video Query Processing

Chih-Chin Liu and **Arbee L. P. Chen***

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
Email : alpchen@cs.nthu.edu.tw

Abstract

In this paper, a video query model based on the content of video and iconic indexing is proposed. We extend the notion of two-dimensional strings to three-dimensional strings (3D-Strings) for representing the spatial and temporal relationships among the symbols in both a video and a video query. The problem of video query processing is then transformed into a problem of three-dimensional pattern matching. To efficiently match the 3D-Strings, a data structure called 3D-List and its related algorithms are proposed. In this approach, the symbols of a video in the video database are retrieved from the video index and organized as a 3D-List according to the 3D-String of the video query. The related algorithms are then applied on the 3D-List to determine whether this video is an answer to the video query. Based on this approach, we have started a project called Vega. In this project, we have implemented a user friendly interface for specifying video queries, a video index tool for constructing the video index, and a video query processor based on the notion of 3D-List. Some experiments are also performed to show the efficiency and effectiveness of the proposed algorithms.

1. Introduction

Many multimedia applications such as World-Wide-Web, video-on-demand, and digital library are getting popular recently. These applications need the support of a video database system to efficiently manage video data and to provide a friendly user interface for the users to retrieve video objects by their *content*. Compared with other media types such as text, image, and audio, video contains richer information[11][21]. However, this richness results in the lack of generally accepted representation of the content of video. Recent proposals regard the representation of the content of video in several ways [11][12][18][20][22]. Based on the object model, Oomoto and Tanaka[18] consider a video object as a sequence of video frames and represent the content of a video object as a collection of attribute/value pairs which are attached to the video object. Weiss et al.[22] view a video object as a three

dimensional box and use algebraic operators to assemble video objects. Smoliar and Zhang [20] model the content of

* To whom all correspondence should be sent.

video objects in two ways. First, according to their topics, video objects are classified into classes and these classes form a tree structure of topical categories. Second, each video shot is represented as a movie icon called a micon which consists of a volume of pixels. By taking a horizontal or vertical slice on the micon, the movement of a symbol in the video object can be traced. Chang et al. [5][6] propose the concept of 2D-string for representing the content of images. In this approach, each object in an image is represented by a symbol and the orders of all symbols along the x-axis and y-axis are stored in two strings. We extend the notion of 2D-string with some modifications to meet the characteristics of video and define 3D-String for the representation of a video query. The problem of video query processing is then transformed into a problem of three-dimensional pattern matching. Many string matching algorithms [1][4][9][13] and pattern matching algorithms [2][3][10] were proposed in the past. However, they are not suitable to apply on the 3D-Strings since the relationships between the symbols in a 3D-String are much more complex than those between symbols in a string or pattern.

In this paper we develop an efficient three-dimensional pattern matching mechanism. To process a video query, we first construct a 3D-String for representing the spatial and temporal relationships between symbols in the video query. Then, the symbol objects of a video object to be evaluated in the video database are retrieved and organized as a 3D-List according to the 3D-String. The 3D-List is a compact graph representation of the spatial-temporal relationships between symbol objects in a video object. Then the 3D-List refinement algorithm is applied on the 3D-List to reduce the number of symbol objects in the 3D-List. Finally, the refined 3D-List is traversed to determine whether the video object is an answer to the video query.

The rest of this paper is organized as follows. Section 2 provides an overview of the Vega video database system implemented at our database laboratory. The function of each component in this system is discussed. Section 3 describes the motivation of our approach for video query processing. The concept of the 3D-Strings for representing video queries and the algorithms for constructing the 3D-String from a video query are discussed in Section 4. The 3D-List data structure and its related algorithms for processing video queries are proposed in Section 5. To show the efficiency and effectiveness of our approach, a series of experiment results are provided in Section 6. Section 7 concludes this paper and describes our future work. Appendix lists the detail of the algorithms for video query processing.

2. Overview of Vega

2.1 Architecture of the Vega Video Database System

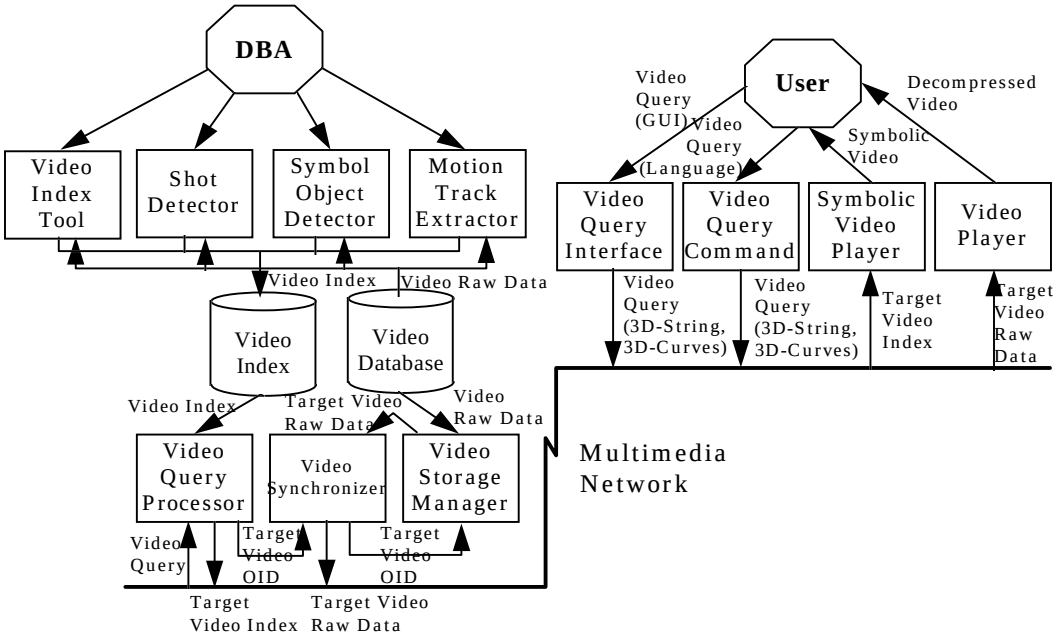


Figure 1: Architecture of the Vega video database system.

Figure 1 illustrates the architecture of the *Vega video database system* which consists of a *video database server* and one or more *video clients*. The video database server provides two basic functions, i.e., the storage management of video data and content-based video retrieval. The first function is supported by the video storage manager (we use EOS [24] object storage manager to manage video objects) and its detail will not be discussed in this paper. The components which provide video indexing, video query interface, and video query processing are described in the following:

• Video Indexing : A video object consists of a sequence of shots in general. Since the relationships between two symbol objects in a shot is semantically stronger than those in two different shots, it is convenient to adopt a shot as a basic index unit. Therefore, a *shot detector* is needed to automatically parse the compressed raw data of a video object into a sequence of shots [14]. A *symbol object detector* is then needed to detect the symbol objects from the shots. As a video object plays, the appearance of each symbol object contained in the video object forms a three-dimensional curve(3D-Curve). This is called the *motion track* of the symbol object. The motion track is one of the major features of a symbol object. A *motion track extractor* traces the motion of a symbol object in continous frames that the

symbol object appears, and encodes and stores the motion track in the video database [7][17]. Since the automatical index derivation techniques are still under investigation, a video index tool is first implemented for manually constructing the video index for video objects. This index tool will be further discussed in Section 2.3.

• *Video Query Interface*: To support content-based video retrieval, a graphical video query interface tool is required for the users to specify video queries. This will be further discussed in Section 2.2. We have also developed a content-based video query language [15]. Users can specify the relationships between any pair of symbol objects as the query conditions using this language.

• *Video Query Processor* : A typical video query processor provides many kinds of feature matching mechanisms. These features depend on which content model the video queries are based. It compares the similarity of a video query and the video index of the video objects and generates target video OIDs as output. Currently, a curve matching method and a 3D-String matching method are implemented in Vega. In traditional databases, the results of a query are sent to the user right after the query is processed. However, since video objects are large, the result of a video query is transmitted to the user site in certain data rate controlled by a *video synchronizer*.

2.2 Video Query Interface

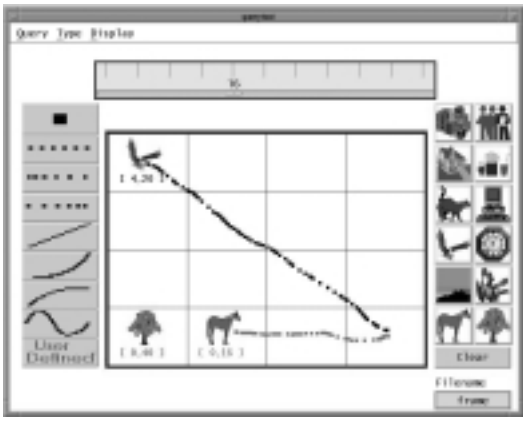


Figure 2: The video query interface.

Figure 2 shows a video query interface we have implemented in Vega. It consists of a video query area, an icon list, a time diagram, and a motion track list. We use an example to illustrate how to use this interface to specify a video query. Assume a user wants to retrieve the video objects that contain a tree located on the lower-left corner of the screen, a horse running from the right side of the tree to the lower-right corner of the screen, and an eagle

flying across the screen from the upper-left corner to the lower-right corner. To specify this video query, a tree icon, a horse icon, and an eagle icon from the icon list are selected and located at the relative places in the video query area. Two motion tracks are then drawn for the horse and the eagle using the functions provided in the motion track list. Finally, a time interval is attached to each icon using the time diagram. This video query is also shown in Figure 2.

2.3 Video Index Tool and Video Index Structures

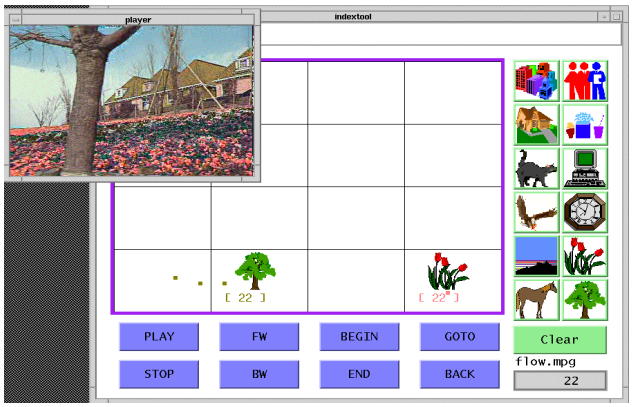


Figure 3: The video index tool.

The video index tool shown in Figure 3 is a graphical interface for building the video index of the video objects. It consists of a video index windows and a video playout window. The video index window consists of a set of VCR buttons, an icon list, and an index display area. It allows users to interactively select interesting symbol objects of a video object and build corresponding indices for these symbol objects. First, the video object to be indexed is selected and played. When an interesting symbol object appears, the user presses the VCR button to pause the playout and chooses an icon representing the symbol object from the icon list and put it at the position that the symbol object appears in the video playout window. The corresponding symbol object will appear in the index display window. Repeat this step for interesting symbol object in this video frame. The object identifiers of the video object, the icon that the symbol object belongs to, and the symbol object itself, and the x, y coordinate values of the central point of the symbol object, and the frame number that the symbol object starts to appear in the video object are stored in a table called *video index table*. Table 1 illustrates an example video index table. The video object V0001 contains ten symbol objects. These symbol objects belong to three icons.

video_oid	***icon_oid***	***symbol_oid***	***x***	***y***	***frame***

V0001	*E*	*S0001*	0	2	4
V0001	*E*	*S0002*	2	0	15
V0001	*E*	*S0003*	1	1	20
V0001	*T*	*S0004*	0	1	1
V0001	*T*	*S0005*	0	2	25
V0001	*T*	*S0006*	2	0	40
V0001	*H*	*S0007*	2	1	1
V0001	*H*	*S0008*	1	1	31
V0001	*H*	*S0009*	2	2	22
V0001	*H*	*S0010*	0	2	40

Table 1: An example video index table.

In addition to the video index table, a video signature is constructed for each video object. Table 2 shows that there are two video objects: V0001 and V0002. The bits in a video signature from left to right represent the existence of the symbol objects which belong to the Icon class, the Tree class (IS-A Icon), the Animal class (IS-A Icon), the Pine class (IS-A Tree), the Olive class (IS-A Tree), the Eagle class (IS-A Animal), and the Horse class (IS-A Animal). Therefore, video object V0001 contains the symbol objects belonging to the Icon class, the Tree class, the Animal class, the Pine class, and the Horse class.

video_oid	***signature***
V0001	*1111001*
V0002	*1010010*

Table 2: A video signature file.

The video signatures can be used to fast preprocess a video query to reduce the number of video objects needed to be further evaluated. When a user specifies a video query using the video interface, a query signature is constructed according to the icons of the video query. Each bit of the query signature denotes whether the corresponding type of symbol objects exists in the video query. The query signature is then compared with every video signature to decide whether the associated video object contains all types of symbol objects in the video query. If it does, the video object needs to be further evaluated to decide whether it is one of the query answers.

3. A Motivative Example

Before we formally describe the concept of the 3D-Strings for representing video queries and the data structure 3D-List for processing video queries, in this section, we use an example to motivate our approach.

Assume a video query Q contains three icons A, B, and C. Icons A and B are at the same place in the x-axis and icon C on the right side of icon B (and icon A). This information can be denoted as the *string* A≡B⇨C. This notation can also be used to represent the relative positions between symbol objects in a video object. For example, assume a video object V has 16 symbol objects $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$, $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{f}_3$ (where $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ belong to icon A, $\mathbf{b}_1$ and $\mathbf{b}_2$ icon B, etc.) and their relative positions are denoted as $\mathbf{a}_1 \equiv \mathbf{e}_2 \equiv \mathbf{b}_2 \Rightarrow \mathbf{f}_1 \equiv \mathbf{d}_1 \equiv \mathbf{f}_2 \Rightarrow \mathbf{a}_3 \equiv \mathbf{b}_1 \equiv \mathbf{a}_2 \equiv \mathbf{c}_1 \equiv \mathbf{d}_2 \Rightarrow \mathbf{e}_1 \Rightarrow \mathbf{c}_2 \equiv \mathbf{d}_3 \equiv \mathbf{c}_3 \Rightarrow \mathbf{f}_3$. We say the video object V is an answer of the video query Q, if V contains three symbol objects $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ such that $\mathbf{a}_i \equiv \mathbf{b}_j \Rightarrow \mathbf{c}_k$.

To decide whether V is an answer of Q, a straightforward method is to match the two associated strings. However, due to the complexity of the relationships between the symbol objects, this method will be very inefficient. For example, when we find $\mathbf{a}_1$ and $\mathbf{b}_2$ match A≡B, we have to check $\Rightarrow \mathbf{f}_1 \equiv \mathbf{d}_1 \equiv \mathbf{f}_2 \Rightarrow \mathbf{a}_3 \equiv \mathbf{b}_1 \equiv \mathbf{a}_2 \equiv \mathbf{c}_1 \equiv \mathbf{d}_2 \Rightarrow \mathbf{e}_1 \Rightarrow \mathbf{c}_2 \equiv \mathbf{d}_3 \equiv \mathbf{c}_3 \Rightarrow \mathbf{f}_3$ to find the symbol object that matchs ⇨C.

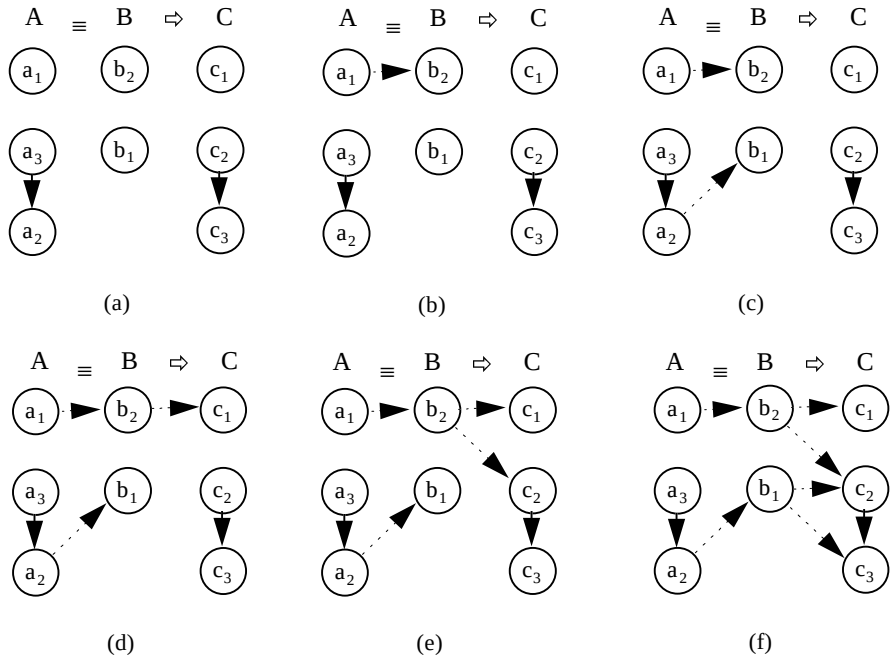


Figure 4: A Motivative Example for Video Query Processing.

Instead of directly matching the two strings, we use the string associated with a video query as a *template* and a data structure to see whether the symbol objects of a video object can fit the template. Only those symbol objects of a video object that belong to the icons of a video query need to be retrieved and checked. In this example, only $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ are retrieved and three sets {$\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$}, {$\mathbf{b}_1$, $\mathbf{b}_2$}, and {$\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$} are formed. The next

step is to check whether there exist three symbol objects selected from {$\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$}, {$\mathbf{b}_1$, $\mathbf{b}_2$}, and {$\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$}, respectively, that match A≡B⇨C. Instead of checking 3*2*3 = 18 combinations of these symbol objects, we develop a data structure to efficiently perform the matching process. First these symbol objects belonging to the same icon are arranged according to their sequence in the string as shown in Figure 4 (a). Then $\mathbf{a}_3$ and $\mathbf{a}_2$ are linked since they are at the same position which implies that they can be treated as a symbol object for the checking. Similarly, $\mathbf{c}_2$ and $\mathbf{c}_3$ are linked. The number of the combinations is reduced to 2*2*2 = 8. Checking the relative position between $\mathbf{a}_1$ and $\mathbf{b}_2$, we find they match A≡B. $\mathbf{a}_1$ and $\mathbf{b}_2$ are thus linked as shown in Figure 4 (b). $\mathbf{a}_1$ and $\mathbf{b}_2$ need not be checked since $\mathbf{b}_2$ and $\mathbf{b}_1$ are not at the same position (they are not linked). Similarly, $\mathbf{a}_3$ and $\mathbf{b}_2$ need not be checked since $\mathbf{a}_3$ and $\mathbf{a}_1$ are not at the same position. Also, $\mathbf{a}_2$ and $\mathbf{b}_2$ need not be checked since $\mathbf{a}_2$ and $\mathbf{a}_3$ are not at the same position. The remaining checking process is as shown in Figure 4 (c) to Figure 4 (f). Totally, only six checkings are needed.

The discussion above only considers the relationships between symbol objects in the x-axis. When we evaluate a video object against a video query, the relationships between the symbol objects in the x-axis, y-axis, and time-axis should all be checked. An efficient algorithm should be developed to combine the one-dimensional results into the three-dimensional results. This is another important problem we have to deal with when we process a video query.

4. The Representation of Video Queries

In this section, we describe the video query model and introduce the notion of 3D-Strings which are used to represent the spatial and temporal relationships between the icons in a video query.

As described in Section 2, the symbol objects which represent the same kind of real world entities are grouped into an icon. The icons in the video database system form an icon hierarchy. Each icon has a graphical notation. A user can operate the query interface tool to select the icons of a video query, to place these icons at some locations in the screen to specify their spatial relationships, and to attach each icon a time interval to specify the period the icon appears. We define the position of an icon by combining the geometrical location and the temporal location of the icon.

Definition 1 *Position of an Icon*

Assume the resolution of the screen of the query interface tool is $X_{max} * Y_{max}$ pixels. The position of an icon in a video query is defined to be a triple (x, y, t), where x and y are the coordinate values in the x-axis and y-axis of the central point of the icon, and t is the number of the frame that the icon begins to appear in the video query.

According to this definition, we only recognize the central point of the icon. The information about the shape or size of the icon are omitted to reduce the complexity of query processing. We will relax this limitation in the future.

Because the resolution of the screen of the query interface tool and the resolution (size) of each video object in the video database can be different, a *uniform resolution model* should be provided as the basis to perform similarity comparison between video queries and the index of video objects. Thus, the screen area is divided into $X_{rank} * Y_{rank}$ grids of equal sizes where X_{rank} and Y_{rank} are user specified. This concept can be extended to the time dimension. The total time intervals of a video query and each video object are both divided into T_{rank} time intervals. The position of an icon in the uniform resolution model, also called the *rank* of the icon, can be defined as follows.

Definition 2 *Rank of an Icon*

Assume the position of an Icon I is (x, y, t), the screen resolution of the query interface tool is $X_{max} * Y_{max}$ pixels and is divided into $X_{rank} * Y_{rank}$ grids, and the total time interval of a video query is set to T_{max} frames and is divided into T_{rank} time intervals. The rank of the icon I is defined to be the triple $(R_x(I), R_y(I), R_t(I))$, where $R_x(I) = \lfloor x * X_{rank} / X_{max} \rfloor$, $R_y(I) = \lfloor y * Y_{rank} / Y_{max} \rfloor$, and $R_t(I) = \lfloor t * T_{rank} / T_{max} \rfloor$.

For any two icons in a video query, basically, there are two kinds of spatial/temporal relationships between them, i.e., the *adjacent relationships* and the *appositional relationships* as defined in the following.

Definition 3 *Adjacent Relationships*

For any two icons I_1 and I_2 in a video query, I_1 is adjacent to I_2 in the x-axis with distance n, denoted by $I_1 \mid_n I_2$, if and only if $R_x(I_1) - R_x(I_2) = n$. Similarly, I_1 is adjacent to I_2 in the y-axis or t-axis with distance n, if and only if $R_y(I_1) - R_y(I_2) = n$ or $R_t(I_1) - R_t(I_2) = n$, respectively.

Definition 4 *Appositional Relationships*

For any two icons I_1 and I_2 in a video query, I_1 is appositional to I_2 in the x-axis, denoted by $I_1 \equiv I_2$, if and only if $R_x(I_1) = R_x(I_2)$. Similarly, I_1 is appositional to I_2 in the y-axis or t-axis, if and only if $R_y(I_1) = R_y(I_2)$ or $R_t(I_1) = R_t(I_2)$, respectively.

The adjacent relationship and the appositional relationship form the *normal spatial-temporal relationship set* $\{|_n, \equiv\}$.

Having defined the adjacent relationships and the appositional relationships, we can further define the *1D-String* and the *3D-String* notation for the representation of the spatial-temporal relationships between icons of a video query.

Definition 5 *Normal 1D-Strings*

A normal 1D-String of length k is a string of the form $I_1\alpha_1I_2\alpha_2 I_3... \alpha_{k-1}I_k$, where each I_i is an icon and each α_j is in $\{|_n, \equiv\}$.

Definition 6 *Normal 3D-Strings*

A normal 3D-String of length k is a triple (X, Y, T), where X, Y, and T are 1D-Strings of the forms $I_1\alpha_1I_2\alpha_2I_3...\alpha_{k-1}I_k$, $I_1'\beta_1I_2'\beta_2I_3'...\beta_{k-1}I_k'$, and $I_1''\gamma_1I_2''\gamma_2I_3''...\gamma_{k-1}I_k''$, respectively. In these strings, each I_i , I_1' , and I_1'' is an icon, each α_j, β_j, and γ_j is in $\{|_n, \equiv\}$, and $\{ I_1, I_2, I_3,..., I_k \} = \{ I_1', I_2', I_3',...,I_k'\} = \{ I_1'', I_2'', I_3'',...,I_k''\}$.

Example 1

Assume A, B, C, and D are four icons. $(A \equiv B |_1 C |_2 D, B |_1 C \equiv A |_2 D, C |_2 D \equiv B \equiv A)$ is a 3D-String, since $\equiv$, $|_1$, and $|_2$ are in$\{|_n, \equiv\}$, and {A, B, C, D} = {B, C, A, D} = {C, D, B, A}. $(A \equiv B |_1 C, B |_1 C\ |_2 D, C |_2 D \equiv A)$ is not a 3D-String, since {A, B, C} ≠ {B, C, D} ≠ {C, D, A}.

The adjacent relationships and the appositional relationships are a straightforward explanation of the spatial-temporal relationships between icons of a video query. There may exist other explanations for them. For example, a user specifies an eagle icon above a tree icon, but he or she may not know how far between them or does not care about their relative position. Therefore, we further define the precedent relationships and the unknown relationships for these situations.

Definition 7 *Precedent Relationships*

For any two icons I_1 and I_2 in a video query, I_1 is precedent to I_2 in the x-axis, denoted $I_1 \Rightarrow I_2$, if and only if $R_x(I_1) < R_x(I_2)$. Similarly, I_1 is precedent to I_2 in the y-axis or t-axis, if and only if $R_y(I_1) < R_y(I_2)$ or $R_t(I_1) < R_t(I_2)$, respectively.

Definition 8 *Unknown Relationships*

For any two icons I_1 and I_2, there is an unknown relationship between I_1 and I_2, denoted I_1 ? I_2, if and only if I_1 and I_2 appear in the same video query and their relative position is unknown.

The adjacent relationships, the appositional relationships, the percedent relationships,

and the unknown relationships form the *extended spatial-temporal relationship set* $\{|_n, \equiv, \Rightarrow, ?\}$.

Definition 9 *Video Query*

A video query Q is a triple (I, R, Q-type), where I is the set of all icons referred to in the query, R is the set of the ranks of icons in I, and Q-type is a number whose value is 0, 1, or 2 denoting the way to translate the spatial-temporal relationships between the icons in I. If Q-type is 0, all spatial-temporal relationships between icons in I are translated into unknown relationships. If Q-type is 1, the adjacent relationships between icons in I are translated into precedent relationships. If Q-type is 2, the adjacent relationships and the appositional relationships between icons in I are retained.

We define the extended 1D-Strings and the extended 3D-Strings for representing video queries.

Definition 10 *Extended 1D-Strings*

An extended 1D-String of length k is a string of the form $I_1\alpha_1I_2\alpha_2 I_3... \alpha_k\text{-}_1I_k$, where each I_i is an icon and each α_j is in $\{|_n, \equiv, \Rightarrow, ?\}$.

Definition 11 *Extended 3D-Strings*

An extended 3D-String of length k is a triple (X, Y, T), where X, Y, and T are 1D-Strings of the forms $I_1\alpha_1I_2\alpha_2I_3...\alpha_k\text{-}_1I_k$, $I_1'\beta_1I_2'\beta_2I_3'...\beta_k\text{-}_1I_k'$, and $I_1''\gamma_1I_2''\gamma_2I_3''...\gamma_k\text{-}_1I_k''$, respectively. In these strings, each I_i , I_1' , and I_1'' is an icon, each α_j, β_j, and γ_j is in $\{|_n, \equiv, \Rightarrow, ?\}$, and $\{ I_1, I_2, I_3,..., I_k \} = \{ I_1', I_2', I_3',...,I_k'\} = \{ I_1'', I_2'', I_3'',...,I_k''\}$.

A video query can be transformed into an extended 3D-String. The transformation takes two steps. First, a video query is transformed into a normal 3D-String according to the ranks of the icons in the video query as stated in Algorithm 1 and Algorithm 2. Second, the normal 3D-String is transformed into an extended 3D-String according to the query type as stated in Algorithm 3 and Algorithm 4.

Algorithm 1 *Build_1D_String(Q, X)*

```
/* input : a video query Q */
/* output : a normal 1D-String */
0:   begin
1:       n ← 1
2:       X ← ϕ
3:       for i = 1 to R_x
4:       begin
5:           if do not exist any icon I with Rx(I) = i
6:               n ← n+1
```

```
7:          else
8:              begin
9:                  pick an icon with Rx(I) = i
10:                 if X = ϕ
11:                     X← I
12:                 else
13:                     X← X + "|n" + I
14:                 end if
15:                 n ← 1
16:                 for each icon I′ with Rx(I′) = i and I′≠ I
17:                     X← X + "≡" + I
18:             end
19:         end if
20:     end
21: end
```

Algorithm 1 constructs a normal 1D-String from a video query in the x-axis. The icons in the video query are sorted according to their Rx values in the rank. If the Rx values of two adjacent icons are the same, an appositional relationship "≡" is inserted between them (as shown in line 17). Otherwise, an adjacent relationship "$|_n$" is inserted (as shown in line 13), where n is the difference of their Rx values. Similarly, this algorithm can be applied in the y-axis or t-axis by changing the Rx values to Ry or Rt values, respectively.

Algorithm 2 *Build_3D_String(Q, (X, Y, T))*

```
/* input : a video query Q */
/* output : a normal 3D-String (X, Y, T) */
0:  begin
1:      Build_1D_String(Q, X)
2:      Build_1D_String(Q, Y)
3:      Build_1D_String(Q, T)
4:  end
```

Algorithm 2 constructs a normal 3D-String from a video query by applying Algorithm 1 in the x-axis, y-axis, and t-axis.

After a normal 3D-String is constructed from a video query, the next step is to change the spatial-temporal relationships in the normal 3D-String according to the query type specified by the user. This can be done by applying the following two algorithms.

Algorithm 3 *Transform_1D_String(X, Q-type)*

```
/* input : a normal 1D-String X = I1α1I2α2 I3... αk-1Ik and the query type Q-type*/
/* output : an extended 1D-String X′ = I1β1I2β2 I3... βk-1Ik */
0:  begin
1:      begin case
2:          case Q-type = 0
```

```
3:              for i = 1 to k-1
4:                  α_i ← "?"
5:          case Q-type = 1
6:              for i = 1 to k-1
7:                  if α_i = "|_n"
8:                      α_i ← "⇨"
9:                  end if
10:     end case
11:     X′ ← X
12:     return X′
13: end
```

Algorithm 4 *Transform_3D_String((X, Y, T), Q-type)*

```
/* input : a normal 3D-String (X, Y, T) and the query type Q-type*/
/* output : an extended 3D-String (X′, Y′, T′) */
0:  begin
1:      X′ ← Transform_1D_String(X, Q-type)
2:      Y′ ←Transform _1D_String(Y, Q-type)
3:      T′ ←Transform _1D_String(T, Q-type)
4:      return (X′, Y′, T′)
5:  end
```

Algorithm 4 constructs an extended 3D-String from a normal 3D-String by applying Algorithm 3 in the x-axis, y-axis, and t-axis. Table 3 summarizes the transformation of the spatial-temporal relationships used in Algorithm 3.

Normal 1D-String	Type 0 Extended 1D-String	Type 1 Extended 1D-String	Type 2 Extended 1D-String
$A \mid_n B$	A ? B	A ⇨ B	$A \mid_n B$
A ≡ B	A ? B	A ≡ B	A ≡ B

Table 3: Transformation rules for normal 1D-Strings.

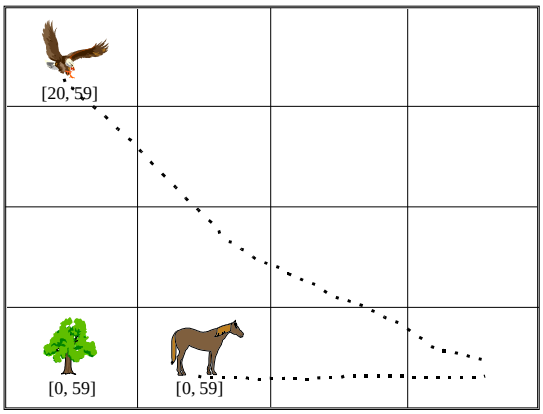


Figure 5: A video query example.

Example 2 Assume there is a video query Q1 as shown in Figure 5, which contains three

icons: an Eagle E, a tree T, and a horse H. The screen resolution of the query interface tool is 1024 * 768 pixels and is divided into 4 * 4 grids, and the total time interval of a video query is set to 60 frames and is divided into 4 time intervals. The positions of E, T, and H are (120, 670, 20), (125, 85, 0), and (384, 110, 0), respectively. The ranks of the three icons are computed as follows.

Rank E = $(R_x(E), R_y(E), R_t(E))$

$= (\lfloor 120*4/1024 \rfloor, \lfloor 670*4/768 \rfloor, \lfloor 20*4/60 \rfloor) = (0, 3, 1)$

Rank T = $(R_x(T), R_y(T), R_t(T))$

$= (\lfloor 125*4/1024 \rfloor, \lfloor 85*4/768 \rfloor, \lfloor 0*4/60 \rfloor) = (0, 0, 0)$

Rank H = $(R_x(H), R_y(H), R_t(H))$

$= (\lfloor 384*4/1024 \rfloor, \lfloor 110*4/768 \rfloor, \lfloor 0*4/60 \rfloor) = (1, 0, 0)$

Applying Algorithm 2, the normal 3D-String representation of Q1 is:

$(X, Y, T) = (E \equiv T \mid_1 H, T \equiv H \mid_3 E, E \mid_1 T \equiv H)$

Applying Algorithm 4, the extended 3D-String representation of Q1 of type 1 is:

$(X', Y', T') = (E \equiv T \Rightarrow H, T \equiv H \Rightarrow E, E \Rightarrow T \equiv H)$

5. Video Query Processing

To check whether the symbol objects of a video object and the spatial and temporal relationships between them satisfies a video query, we introduce a data structure called 3D-List and its related algorithms in this section. First, we will describe the structure and algorithms of 1D-List, the one-dimensional special case of 3D-List. The algorithms for combining and refining the three one-dimensional results are then proposed. Finally, the algorithm for generating the three-dimensional final results will be discussed.

5.1 Generating 1D Results

As we have described in the previous section, a video query can be represented as a 3D-String. The video objects which contain one or more sets of symbol objects that *match* the 3D-String of a video query are the answers of the video query. Since a 3D-String consists of three 1D-Strings, we first define the *solutions* of 1D-String. Let Icon() be a function which returns the corresponding icon of a symbol object.

Definition 12 *Solutions and Solution Set of a 1D-String*

Assume there is a 1D-String X of length k where $X = I_1\alpha_1I_2\alpha_2\ldots\alpha_{k-1}I_k$. A set of k symbol objects $S = \{s_1, s_2, \ldots, s_k\}$ is a solution of X, if and only if for each symbol s_i, $Icon(s_i) = I_i$, $1 \le i \le$

k, and $s_1\alpha_1 s_2\alpha_2 \ldots \alpha_{k-1}s_k$ holds. The solution set of the 1D-String X is the set of all solutions of X.

Our approach for video query processing is based on the notion of the equivalence relationships. Equivalence relationships are defined to reduce the complexity of the representation of the relationships between objects. For example, assume there are five numbers 1, 2, 3, 6, and 7 and six great-than relationships needed to be recorded between them: 1 < 6, 1 < 7, 2 < 6, 2 < 7, 3 < 6, and 3 < 7. These great-than relationships can be more concisely represented as {1, 2, 3} < {6, 7} where {1, 2, 3} and {6, 7} are two *equivalence sets* and there is an *equivalence relationship* between each pair of numbers in an equivalence set. We define equivalence relationships as follows.

Definition 13 *Complete Solution Set, Equivalence Set, and Equivalence Relationships*

A *complete solution set* S of a 1D-String X of length k where $X = I_1\alpha_1 I_2\alpha_2 \ldots \alpha_{k-1}I_k$ is a solution set $\{\{s_1, s_2, ..., s_k\} \mid s_1 \in S_1, s_2 \in S_2, \ldots, s_k \in S_k\}$ of X, where S_i is a set of symbol objects and $Icon(s) = I_i$ for each $s \in S_i$, $1 \le i \le k$, and for *each* $s_1 \in S_1, s_2 \in S_2, \ldots, s_k \in S_k$, $s_1\alpha_1 s_2\alpha_2 \ldots \alpha_{k-1}s_k$ holds. Each set of symbol objects S_i is called an *equivalence set* of X and every pair of symbol objects in an equivalence set has an *equivalence relationship* between them.

Definition 14 *Equivalence Group of S with respect to* $\alpha_{i-1}I_i\alpha_i$

Assume there is a 1D-String $X = I_1\alpha_1 I_2\alpha_2 \ldots \alpha_{k-1}I_k$ and a set of symbol objects S. Let α_0 and α_k be "⇨". The ordered list of symbol objects $G = <s_1, s_2, \ldots, s_n>$, $Icon(s_j) = I_i$, $1 \le j \le n$, is an equivalence group of S with respect to $\alpha_{i-1}I_i\alpha_i$, if and only if for any two symbol objects s_1 and s_2 of G and any two symbol objects s_1' and s_2', if $s_1' \alpha_{i-1} s_1 \alpha_i s_2'$ holds, $s_1' \alpha_{i-1} s_2 \alpha_i s_2'$ holds. For every pair of symbol objects in an equivalence group with respect to $\alpha_{i-1}I_i\alpha_i$, there is an *equivalence relationship (with respect to* $\alpha_{i-1}I_i\alpha_i$*)* between them. These equivalence relationships are denoted as ε. For example, the equivalence group $G = <s_1, s_2, \ldots, s_n>$ can be denoted as $s_1 \varepsilon s_2 \varepsilon \ldots \varepsilon s_n$.

We use the video object shown in Table 1 to illustrate the concept of the equivalence group. Assume there is a 1D-String X = E≡T⇨H. The ordered list of symbol objects G = <s0004, s0005> is an equivalence group with respect to ≡T⇨, since Icon(s0004) = Icon(s0005) = T and the two symbol objects have the same R_x value which means for any two symbol objects s_1' and s_2', if $s_1' \equiv$ s0004 ⇨ s_2' holds, $s_1' \equiv$ s0005 ⇨ s_2' holds.

Except for the first icon and the last icon, there are two extended spatial-temporal relationships an icon I_i can be involved in a 1D-String, i.e., α_{i-1} and α_i. Since there are four

types of extended spatial-temporal relationships, there are 4 * 4 = 16 cases we should consider when we build the equivalence groups with respect to $\alpha_{i-1}I_i\alpha_i$. Since the unknown relationship and the other types of extended spatial-temporal relationships never appear together in the same 1D-String, the number of cases reduces to 3 * 3 = 9. Further, the adjacent relationships and the appositional relationships follow the same rules to construct the equivalence groups, the number of cases reduces to 2 * 2 = 4. By analyzing the four cases, the following rule is derived to link the symbol objects with equivalence relationships to form equivalence groups.

Assume there is a 1D-String $X = I_1\alpha_1I_2\alpha_2...\alpha_{k-1}I_k$ and a set of symbol objects S. Let α_0 and α_k be "⇨". For each icon I_i, if $\alpha_{i-1} = \alpha_i$ = "⇨", then the equivalence group of S with respect to $\alpha_{i-1}I_i\alpha_i$ is the ordered list of n symbols $G = < s_1, s_2, ..., s_n >$ where $s_1, s_2, ..., s_n$ are symbols in S, $Icon(s_j) = I_i$ for $1 \le j \le n$, and $R_x(s_j) \le R_x(s_{j+1})$ for $1 \le j < n$ (Note that $R_x(s_i)$ represents the R_x value of s_i in the rank); otherwise, symbols s_j, $Icon(s_j) = I_i$, with the same R_x value will form an equivalence group.

Note that, if $\alpha_{i-1} = \alpha_i$ = "⇨", there is only one equivalence group of S with respect to $\alpha_{i-1}I_i\alpha_i$; otherwise, the number of equivalence groups of S with respect to $\alpha_{i-1}I_i\alpha_i$ is the number of different R_x values of all symbol objects in I_i.

Having defined the extended spatial-temporal relationships and the equivalence relationships with respect to $\alpha_{i-1}I_i\alpha_i$, now we can define 1D-List, the data structure used to perform 1D-String matching.

Definition 15 *1D-List*

A 1D-List is a 5-tuple (X, S, $\{s_{start}, s_{end}\}$, R, E), where X is a 1D-String, S is a set of symbol objects, R is the set of relationships among S over the extended relationship set $\{|_n, \equiv, ⇨, ?\}$, and E is the set of equivalence relationships among the symbol objects of the equivalence groups of S. s_{start} and s_{end} are two dummy symbol objects defined to mark the start point and end point of the 1D-List.

Figure 6 (d) shows a 1D-List example. The 1D-String X = E ≡ T ⇨ H. The symbol object S = {s0001, s0002, s0003, ..., s0010} forms six equivalence groups {s0001}, {s0002}, {s0003}, {s0004, s0005}, {s0006}, and {s0007, s0008, s0009, s0010}. The equivalence relationship with respect to ≡ T ⇨ is s0004 ε s0005. The three equivalence relationships with respect to ⇨ H ⇨ is s0010 ε s0008, s0008 ε s0007, and s0007 ε s0009. The extended spatial-temporal relationships between S are denoted as dashed arrow in the Figure.

The goal of the construction of the 1D-List is to find the solutions of its associated 1D-String. However, a 1D-List may contain symbol objects that are not involved in any solutions of the 1D-String. Therefore, the 1D-List should be refined to remove these redundant symbol objects.

Definition 16 *Path*

Let α_0 and α_k be "⇨". A path of a 1D-List L = (X, S, {s_{start}, s_{end}}, R, E) is $s_{start}\alpha_0 s_1\alpha_1 s_2\alpha_2 \ldots \alpha_{k-1} s_k \alpha_k s_{end}$ where si is in S for $1 \le i \le k$, and α_i is in $R \cup E$ for $1 \le i \le k-1$.

From a path of a 1D-List a complete solution set of the associated 1D-String can be directly derived.

Having defined the necessary terms for the 1D-List, now we describe how to find the solution of a 1D-String using 1D-Lists. The construction of the 1D-List for the 1D-String $X = I_1\alpha_1 I_2\alpha_2 \ldots \alpha_{k-1} I_k$ and the video object V goes through the following steps(Algorithm 5 in the Appendix):

Step 1: Retrieve every symbol object s and its rank $R_x(s)$ from the video index where s is in V and Icon(s) = I_i. For each icon I_i, we store the symbol objects that belong to it as S_i. Let us use an example to illustrate the construction. Assume the video object V is as shown in Table 2 and the extended 1D-String X is E ≡ T ⇨ H. The symbol objects s0001, s0002, s0003, ..., s0010 are retrieved from Table 2 and form three symbol object sets S_1 = {s0001, s0002, s0003}, S_2 = {s0004, s0005, s0006}, and S_3 = {s0007, s0008, s0009, s0010} as shown in Figure 6 (a).

Step 2: Sort the symbol objects in each symbol object set S_i according to their rank values. In this example, the three symbol object sets are sorted into S_1 = {s0001, s0003, s0002}, S_2 = {s0004, s0005, s0006}, and S_3 = {s0010, s0008, s0007, s0009} as shown in Figure 6 (b).

Step 3: Build the equivalence relationships between the symbol objects in each symbol object set S_i, for $1 \le i \le k$. If the two spatial-temporal relationship α_{i-1} and α_i are both "⇨", all symbol objects in S_i are linked into an equivalence group. Otherwise, the symbol objects with the same R_x value in S_i are linked to form an equivalence group. In this example, S_1 has three equivalence groups {s0001}, {s0002}, and {s0003}, since α_1 = "≡" ≠ "⇨" and s0001, s0002, and s0003 all have different R_x values. S_3 has only one equivalence group {s0010, s0008, s0007, s0009} since both α_2 and α_3 are "⇨" as shown in Figure 6 (c). This algorithm for constructing the

equivalence relationships is stated in Algorithm 6 in the Appendix.

Step 4: Build the spatial-temporal relationships between each pair of sequential symbol object sets S_i and S_{i+1}, for $1 \leq i \leq k-1$. This construction is based on the distribution property of the spatial-temporal relationship to the equivalence relationship. It can be denoted as follows: let α be an spatial-temporal relationship, ε be an equivalence relationship, s_1, s_2, s_3, and s_4 be four symbol objects. Then $s_1 \varepsilon s_2 \alpha s_3 \varepsilon s_4$ if and only if $s_1 \alpha s_3$, $s_2 \alpha s_3$, $s_1 \alpha s_4$, and $s_2 \alpha s_4$. The formal algorithm for the construction of the spatial-temporal relationships is described in Algorithm 7 in the Appendix. In this example, three spatial-temporal relationships are constructed. They are s0001 α s0004, s0002 α s0006, and s0005 α s0008 as shown in Figure 6 (d).

Now we have constructed a 1D-List for the video object V with respect to the 1D-String X. The next step we have to do is to remove the symbol objects not included in any solution of X. These symbol objects are also the symbol objects that do not appear in any path of the 1D-List. For each k equivalence groups G_1, G_2, ..., G_k of the symbol object sets S_1, S_2, ...,S_k, respectively, if there exists a path in these equivalence groups, the symbol objects above the uppermost path or below the lowest path in these equivalence groups can be removed since these symbol objects do not appear in any path of the 1D-List.

Step 5: Forward remove the equivalence relationships above the uppermost path of the equivalence groups. In this example, the uppermost path is s0001 α s0004 ε s0005 α s0008 ε s0007 ε s0009 and s0010 is removed as shown in Figure 6 (e).

Step 6: Backward remove the equivalence relationships below the lowest path of the equivalence groups. In this example, the lowest path is also s0001 α s0004 ε s0005 α s0008 ε s0007 ε s0009. Three symbol objects s0003, s0002, and s0006 are removed since they do not belong to any path as shown in Figure 6 (f). Now, we get the 1D-List representation of the solutions of the extended 1D-String X = E $\equiv$ T $\Rightarrow$ H. Traverse the 1D-List from s_{start} to s_{end}, the path that contains the solutions is s0001 α s0004 ε s0005 α s0008 ε s0007 ε s0009. The six solutions are: {s0001, s0004, s0007}, {s0001, s0004, s0008}, {s0001, s0004, s0009}, {s0001, s0005, s0007}, {s0001, s0005, s0008}, and {s0001, s0005, s0009}.

The algorithm performs forward and backward removing is stated in Algorithm 8 in the Appendix.

5.2 Combining and Refining the 1D Results

In this subsection, we discuss how to combine the three one-dimensional results of 1D-Strings. First we define the solution of a 3D-String:

Definition 17 *Solutions and Solution Set of a 3D-String*

Assume there is a 3D-String (X, Y, T) of length k where $X = I_1\alpha_1I_2\alpha_2\ldots\alpha_{k-1}I_k$, $Y = I_1'\beta_1I_2'\beta_2I_3'\ldots\beta_{k-1}I_k'$, and $T = I_1''\gamma_1I_2''\gamma_2I_3''\ldots\gamma_{k-1}I_k''$. A set of k symbol objects $S = \{s_1, s_2, \ldots, s_k\}$ is a solution of (X, Y, T), if and only if there exist solutions S_1, S_2, and S_3 of 1D-Strings X, Y, and T, respectively, such that $S = S_1 = S_2 = S_3$. The solution set of the 3D-String (X, Y, T) is the set of all solutions of (X, Y, T).

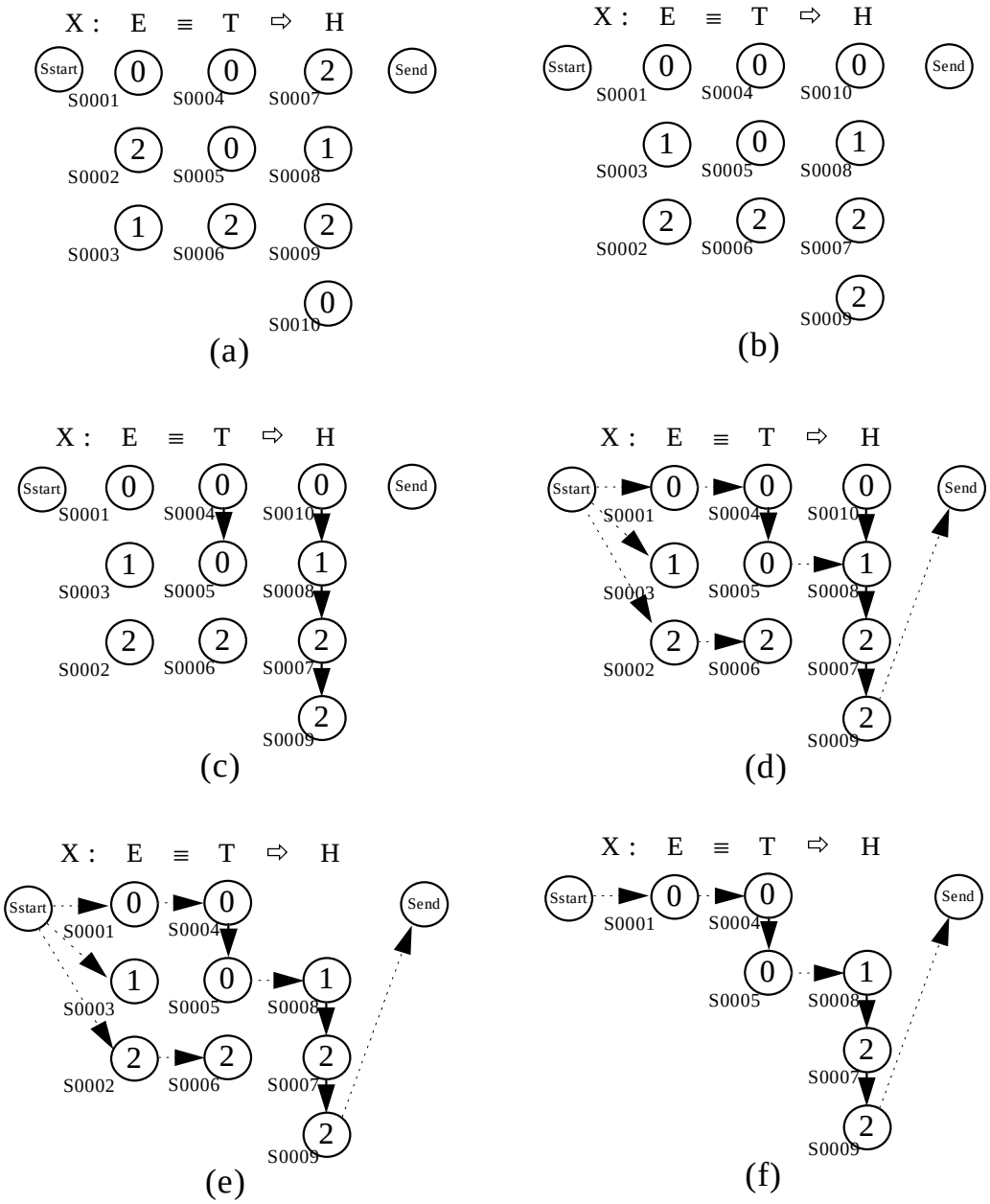


Figure 6: A 1D-List example.

The concept of the complete solution sets of a 1D-String is extended to the 3D-String:

Definition 18 *Complete Solution Sets of a 3D-String*

A *complete solution set* of a 3D-String (X, Y, T) of length k is the solution set S of (X, Y, T) and S is also the complete solution set of each of the 1D-Strings X, Y, and T.

Definition 19 *3D-List*

A 3D-List L consists of 1D-Lists L_x, L_y, and L_t where L_x = (X, S, {s_{start}, s_{end}}, R_x, E_x), L_y =(Y, S, {s_{start}, s_{end}}, R_y, E_y), L_t =(T, S, {s_{start}, s_{end}}, R_t, E_t), and (X, Y, T) is a 3D-String.

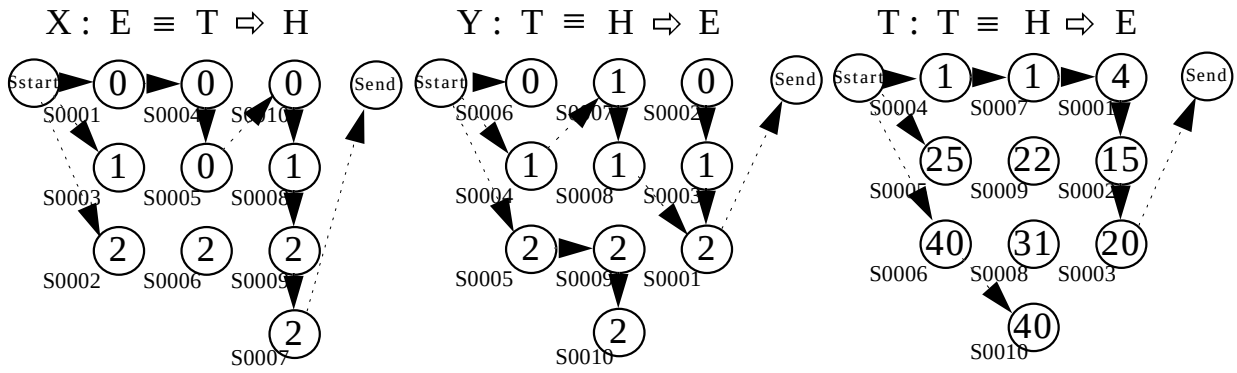
The construction of the 3D-List for a video query Q and a video object V goes as follows: First, we transform the video query Q into the corresponding 3D-String (X, Y, T). Then three 1D-Lists are constructed with respect to the three 1D-Strings X, Y, and T and the video object V (Algorithm 9 in the Appendix). The three 1D-Lists compose a 3D-List since (X, Y, T) is a 3D-String.

Example 3 Build the 3D-List for the 3D-String (X, Y, T) of the video query Q1 (assume the type of query is 1) in Figure 5 and the video object whose index is shown in Table 2.

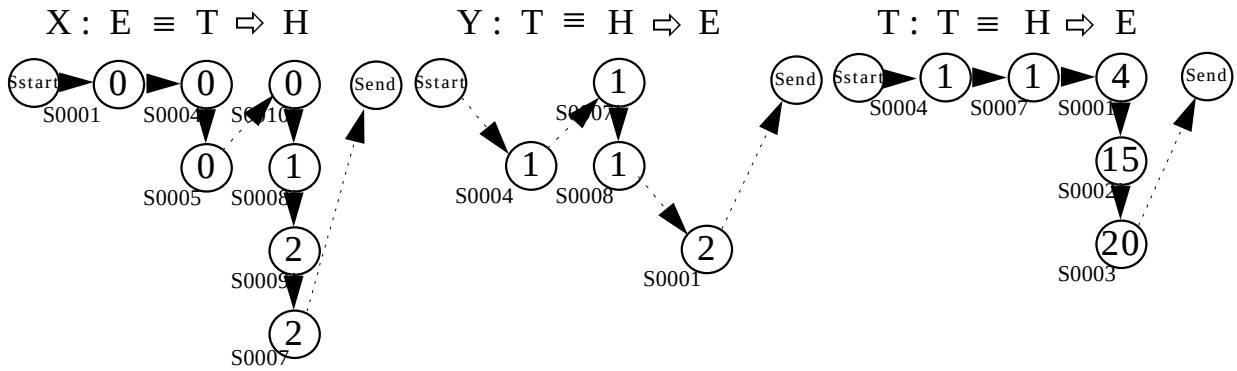
First, we transform the video query Q1 into the extended 3D-String (X, Y, T) = (E≡T⇨H, T≡H⇨E, T≡H⇨E).

Then, we apply 1D-List construction algorithm three times, the 3D-List is constructed as shown in Figure 7 (a).

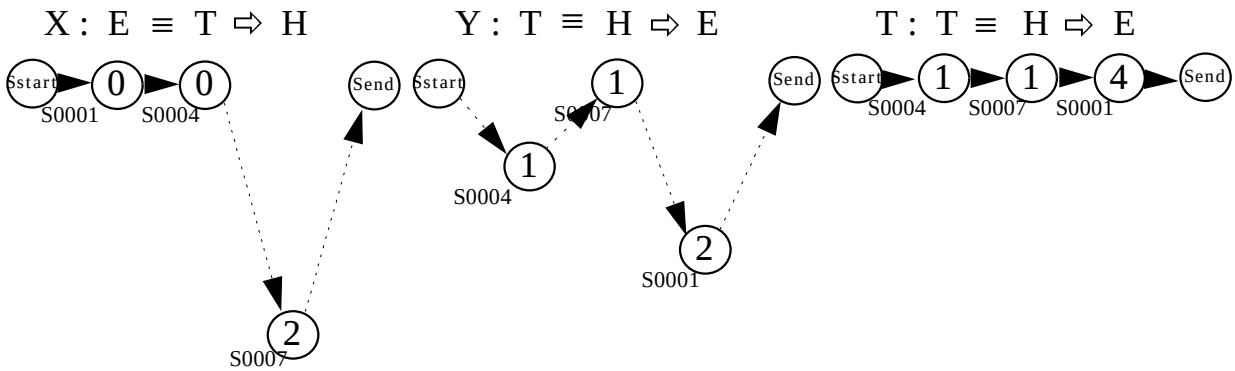
Checking the 3D-List shown in Figure 7 (a), we find that there are many symbol objects that are not contained in any path of one of the three 1D-Lists. Moreover, even if a symbol object is contained in a path in a 1D-List, it may not be contained in the other two 1D-Lists. Before we apply the algorithm to generate the solutions of the 3D-List, which will be discussed in the next subsection, we should remove the symbol objects that are not included in any solution of the 3D-String of the video query as much as possible. This can reduce the number of symbol objects to be checked. We call this removing process as *3D-List refinement*. We propose an algorithm to perform the 3D-List refinement. This refinement algorithm uses the forward and backward removing algorithm proposed in the previous subsection on each 1D-List and lets the new symbol objects sets of the resultant 1D-Lists be the intersection of the symbol objects sets of the resultant 1D-Lists. Then, this process is repeatedly applied until no more redundant symbol objects can be removed. This algorithm is stated in Algorithm 10 in the Appendix. We use the following example to illustrate the 3D-List refinement algorithm.



(a)



(b)



(c)

{S0001, S0004, S0007}

(d)

Figure 7: A 3D-List refinement example.

Example 4 Assume the 3D-List constructed in Example 3 is to be refined using the 3D-

List refinement algorithm. First, we apply the forward and backward removing algorithm on each 1D-List. The result is shown in Figure 7 (b). The intersection of the three symbol object sets in the resultant 1D-Lists is {s0001, s0004, s0007}. Adjusting the three 1D-Lists, the result is shown in Figure 7 (c) and no more symbol objects can be removed by the forward and backward removing algorithm. Therefore, the 3D-List shown in Figure 3(c) is the result of the 3D-List refinement algorithm.

5.3 Generating the 3D Results

By applying one of the two refinement algorithms described in the previous subsection we get a refined 3D-List. However, to find the final solutions of the 3D-String of a video query, we still have to traverse the refined 3D-List to generate a set of paths in each 1D-List. Assume paths P_1, P_2 and P_3 are generated from the 1D-Lists L_x, L_y, and L_t, respectively. By intersecting the symbol object sets of P_1, P_2 and P_3, we can decide whether the combination of P_1, P_2 and P_3 contains the final solutions. If every icon in the video query has one or more corresponding symbol objects in the intersection result, from the intersection result a complete solution set of the 3D-String can be directly derived. That is, the decision depends on whether the total number of different icons associated with the intersection result equals to the number of icons in the 3D-String. This algorithm is stated in Algorithm 11 in the Appendix.

We use the result of Example 4 to illustrate the generating process. In Example 4, each 1D-List has only one path. For L_x, the path is s0001 α s0004 ε s0005 α s0010 ε s0008 ε s0009 ε s0007, for L_y, the path is s0004 α s0007 ε s0008 α s0001, and for L_t, the path is s0004 α s0007 α s0001 ε s0002 ε s0003. Intersecting the three symbol sets of the paths we get: S = {s0001, s0004, s0005, s0010, s0008, s0009, s0007}$\cap${s0004, s0007, s0008, s0001}$\cap${s0004, s0007, s0001, s0002, s0003} = {s0004, s0007, s0001}. Since Icon(s0004) = T, Icon(s0007) = H, and Icon(s0001) = E, the total number of different icons of S is three which is equal to the number of icons in the 3D-String. Therefore {s0004, s0007, s0001} is a solution of the 3D-String.

6. Performance Analysis

To show the efficiency and effectiveness of our video query processing techniques, we perform a series of experiments which can be classified into two groups. The first group of experiments is made on the synthesized video indexes. There are five cost factors

dominating the performance of the video query processing algorithms: the number of icons in the video queries, the number of video objects, the total number of different icons in the video database, the average number of symbol objects in a video object, and the maximum rank in each axis. We can freely set the values of the five cost factors in the synthesized video indexes. Since the string matching algorithms proposed in the past are not suitable to apply on the 3D-Strings, we use a *direct string matching* algorithm for the comparison. The direct string matching algorithm goes quite simple: to match an extended 1D-String $I_1\alpha_1I_2\alpha_2I_3...\alpha_{k-1}I_k$ with a set of n ordered symbol objects $S = [s_1, s_2, ..., s_n]$ (sorted according to their ranks) in a video object, we select the first k symbol objects from S. Then each symbol object selected is checked one by one from the right. If a selected symbol object s_i matches (i.e. $Icon(s_i) = I_i$ and $s_{i-1}\alpha_1 s_i$ holds), the next symbol object will be checked. If s_i does not match and s_{i+1} is not selected, s_i is replaced with s_{i+1} and s_{i+1} will be checked. If s_i does not match and s_{i+1} is selected, the checking process will go back to the previous selected symbol object. This process continues until the first selected is matched (which means the video object is an answer of the 1D-String) or no more symbol object left while the first selected is still not matched (which means the video object is not an answer of the 1D-String).

The second group of experiments is made on 200 real video objects. Each video object is a MPEG video clip about one minute long. The symbol objects in each video object are specified by the DBA using the video index tool. All algorithms are implemented on a Sun Sparc20 workstation with SunOS 4.1.4.

6.1 Video Index Construction

In our approach, a preprocessing phase is needed to construct the video index for all video objects in the video database. The first experiment shows the index construction cost. Since the video index construction algorithm is not necessary for the synthesized video indexes, we use real video objects to perform this experiment. We measure the execution time versus the number of symbol objects. As illustrated in Figure 8, the index construction cost is linear to the number of symbol objects. Since video index construction can be done off-line, we do not add this cost to the execution time of the following 3D-String matching experiments.

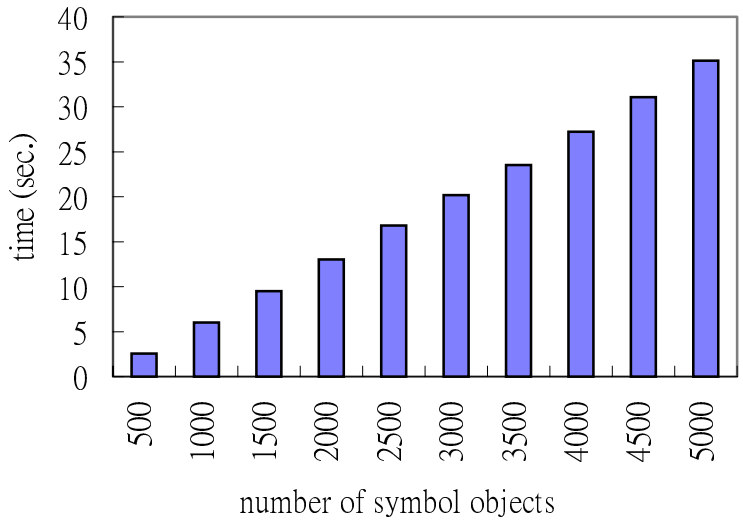


Figure 8: Indexing time vs. number of symbol objects.

6.2 Synthesized Video Indexes

In this subsection, we show the efficiency of our 3D-List matching algorithms and compare with the direct string matching approach.

The execution cost of every experiment is measured by the average elapsed time of 100 video queries. Since the icons in the video queries are specified by users, we assume the number of icons to be in the range between 1 and 10. We generate the video index for 1000 video objects. For each video object, we assign randomly from 100 to 1000 symbol objects to it. The total number of different icons in the video database is set in the range between 100 and 500. The maximum rank in each axis is set in the range between 2 and 10. Based on these synthesized video index, we perform five experiments. In each experiment we change one cost factor and fix the other four cost factors. The values we choose for the fixed cost factors are 4 icons in a video query, 200 video objects, 100 symbol objects in a video object, 128 different icons in the video database, and the maximum rank in each axis is 4. The experiment results are shown as follows.

Figure 9 illustrates the execution time versus the number of icons in the video query of type 0, type 1, and type 2. Since for video queries of type 0, the 3D-List approach only needs to check whether there exist corresponding symbol objects in a video object for each icon in the video query, the least execution time is required. In all cases, compared with the direct string matching algorithms, the 3D-List algorithms only need 1/6 execution time to process the video queries. Video queries of type 2 take more execution time since more relationships have to be checked. Note that for the 3D-List approach, the execution time becomes smaller

when the number of icons in the video queries is larger than seven. This is because there does not exist an answer in the video database for these video queries.

Figure 10 illustrates the execution time versus the number of video objects in the video database. The execution time grows linear as the number of video objects increases for both approaches in all three types of video queries. This may not be acceptable for very large video databases. However, we can implement video signature for the video index to reduce the number of video objects needed to be checked.

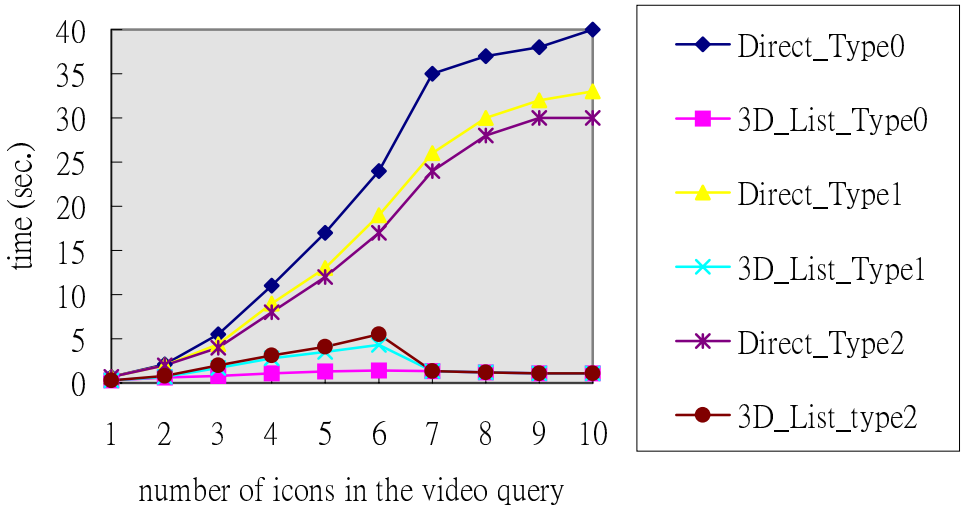


Figure 9: Execution time vs. number of icons in the video query.

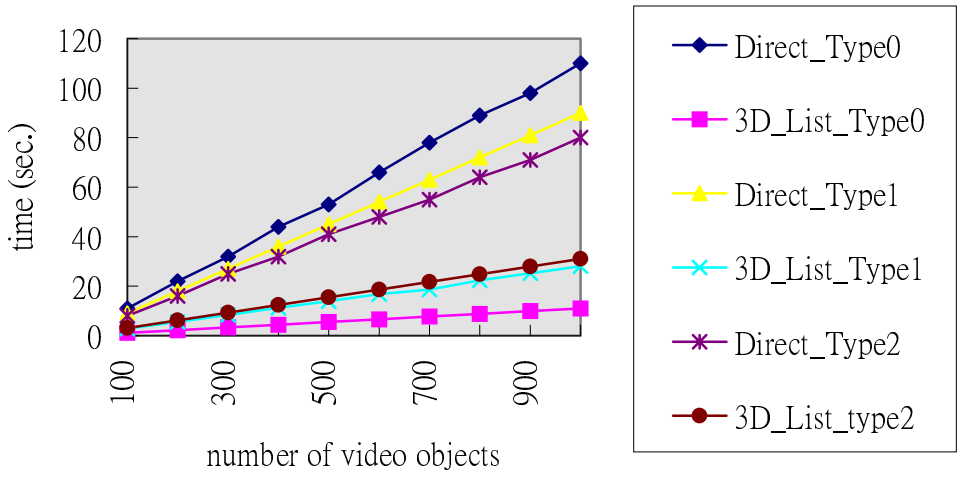


Figure 10: Execution time vs. number of video objects.

Figure 11 illustrates the execution time versus total number of different icons in the video database. Both approaches benefit when the total number of different icons increases.

However, the execution time of the 3D-List approach reduces faster since the number of symbol objects retrieved from the video index is reverse proportional to the total number of different icons in the video database. Also note that when the total number of different icons is larger than 300, it is very possible that there does not exist an answer in the video database for these video queries. In this case the 3D-List algorithms terminate quickly.

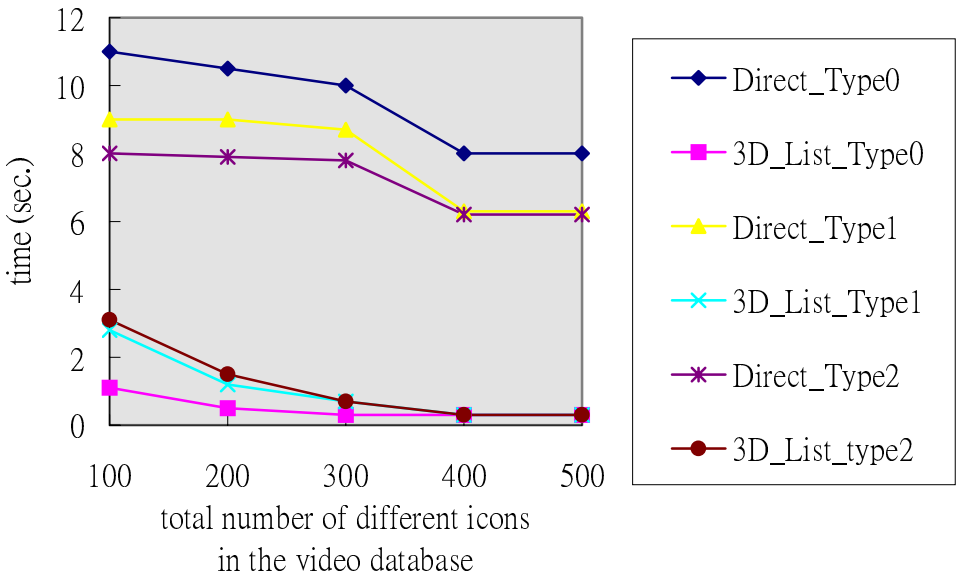


Figure 11: Execution time vs. total number of different icons in the video database.

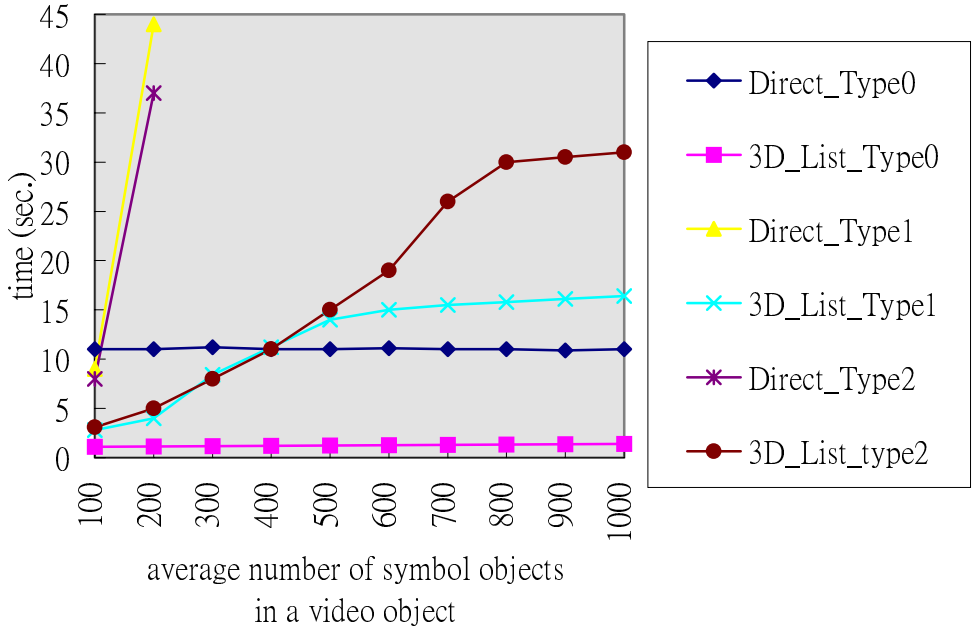


Figure 12: Execution time vs. average number of symbol objects in a video object.

Figure 12 illustrates the execution time versus average number of symbol objects in a video object. For each video query of type 0, since both algorithms will terminate when they find an answer of the video query, the execution time increases slowly as the average number of symbol objects in a video object increases. However, for video queries of type 1 and type 2, the execution time of the direct string matching algorithms increase sharply since they have to check most combinations of the symbol objects in each video object. On the other hand, since the extra time needed to spend for the 3D-List algorithms is to retrieve the increased symbol objects from the video index and to check the relationships between them, the execution time grows slowly.

Figure 13 illustrates the execution time versus maximum rank in each axis. For video queries of type 0, since the answers are independent of the ranks of the symbol objects, the execution time for both algorithms remain the same as the maximum rank in each axis increases. For video queries of type1 and type 2, the execution time of the direct string matching algorithms decreases as the maximum rank in each axis increases. This is because less combinations of symbol objects in a video objects are needed to be checked. For video queries of type 1 and type 2, the execution time of the 3D-List algorithms increases as the maximum rank in each axis increases since the number of paths in each 1D-List increases. However, for video queries of type 2, when the maximum rank in each axis is

large (larger than 7 in this case), it is very possible that there does not exist an answer in the video database. Therefore, the execution time is reduced.

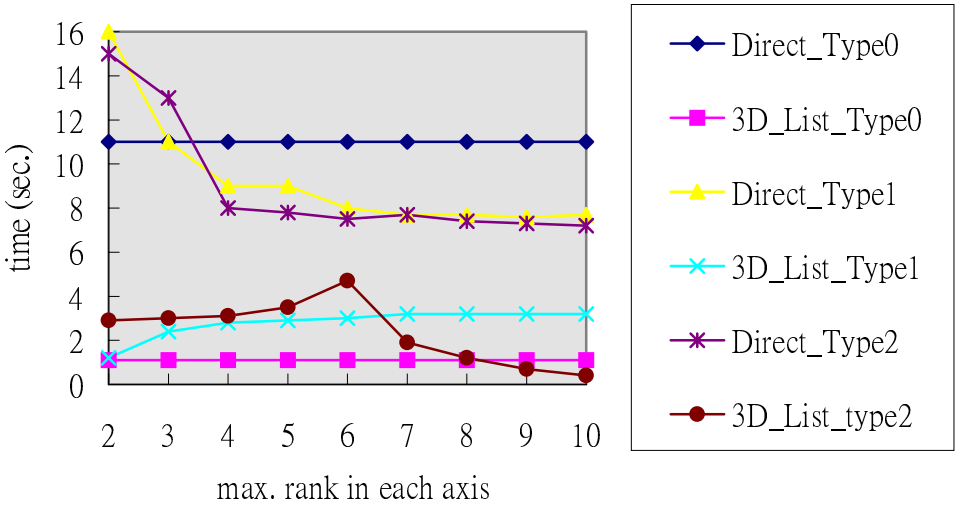


Figure 13: Execution time vs. maximum rank in each axis.

6.3 Real Video Data

In this subsection, we show the effectiveness of our video query processing techniques. Since the performance of both indexing and query processing algorithms depends on the number of symbol objects, the first thing we want to know is how many symbol objects are contained in a video object in average. In our example video database, there are 70 video objects of wild animals, 40 video objects of cartoons, 40 video objects of movies, and 50 video objects of TV news. All video objects are one minute long. In general, we specify five symbol objects from each frame in a video object. Typically, a video object of one minute long contains 1800 frames. To represent the movements of symbol objects, at least a frame should be indexed for every 10 frames. Thus, the average number of symbol objects in a video object is about 900 (883 in our example video database).

The second measurement is the number of icons in the video database. We found for different areas of video objects, for example, wild animals and cartoons, there are few icons in common. Therefore, designing an icon hierarchy for every area of video data is suggested. In our sample database, we design four specialized icon hierarchies with 48 icons and a general icon hierarchy with 64 icons. Thus the total number of icons is 256.

For real video databases, the clustering effect of video objects is very apparent, i.e., two video objects of the same area tend to contain similar symbol objects. Therefore, if we pose a

video query with two icons which are selected from two specialized icon hierarchies, the query will return no answer in general. On the other hand, if we pose a video query with a person icon, almost all video objects satisfy this query (except for video objects of wild animals). According to our experiment, the execution time of every video query is smaller than 3 seconds, no matter now many icons are specified in the video queries.

7. Conclusion

This paper discusses a methodology for the indexing and retrieval of video objects. Based on the well known representation of images — the 2D-String, we define the notion of 3D-String for the representation of the spatial and temporal relationships between icons in a video query. Based on the 3D-Strings, the problem of video query processing is transformed into a problem of three-dimensional pattern matching. Since the string matching algorithms proposed in the past cannot solve this problem, the 3D-List data structure and its related algorithms are proposed. There are two major techniques involved in the proposed algorithms. First, the video index enables us to retrieve only the symbol objects which are related to the icons involved in the video queries for further processing. Second, the equivalence groups constructed for two sets of symbol objects based on the notion of equivalence relationships prevent us from exhaustively checking every pair of symbol objects in the two sets of symbol objects.

To show the efficiency and effectiveness of the proposed algorithms, we perform a series of experiments on the synthesized video indexes in which the influence of the five major cost factors is illustrated. The time complexity is shown to be equal to or smaller than O(n) for each cost factor. Compared with the direct string matching algorithm, more than 80% query processing speedup is achieved. For the real video database, our experiments show that the execution time of each video query is smaller than three seconds.

Video data modeling has been an active research topic. The related work can be classified according to whether the symbol objects in the video objects are modeled. Our video data model can strengthen the video data models without the notion of symbol objects [11][12][18][20][21][22]. Since the notion of 3D-String in the video databases is an extension of 2D-String in the image databases, the 3D-List data structure and its related algorithms can be easily applied to image or video queries based on the notion of symbol objects [5][6][15][25][26].

We are currently working on extending the proposed methodology in many ways. First, we will develop an index structure for the video signature and the video index to avoid exhaustively searching all video objects in the video database. Second, we are developing a

curve matching scheme for the motion tracks of symbol objects [27]. An automatic symbol object and motion track detection technique is also under development. Finally, the video query processing and synchronization mechanisms in the distributed environment will be investigated.

References:

[1] Aho, A. V. and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Comm. ACM*, Vol. 18, pp. 333-340, June 1979.

[2] Baker, T. P., "A Technique for Extending Rapid Exact-Match String matching to Arrays of More Than One Dimension," *SIAM J. Comput.* Vol. 7, pp. 533-541, Nov. 1978.

[3] Bird, R. S., "Two Dimensional Pattern Matching," *Information Processing Letters*, Vol. 6, No. 5, pp. 168-170, Oct. 1977.

[4] Boyer, R. S. and J. S. Moore, "A Fast String Searching Algorithm," *Comm. ACM*, Vol. 20, pp. 762-772, Oct. 1977.

[5] Chang, S. K., Q. Y. Shi, and C. W. Yan, "Iconic Indexing by 2-D Strings," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No. 3, pp. 413-428, May 1987.

[6] Chang, S. K., C. W. Yan, D. C. Dimitroff, and T. Arndt, "An Intelligent Image Database System," *IEEE Trans. on Software Eng.*, Vol. 14, No. 5, pp.681-688, May 1988.

[7] Chen, A. L. P., C. C. Liu, K. L. Lee, and C. C. Chen, "The Design of a Video Database System," *Proc. of Real-Time and Media Systems Conf.*, 1995.

[8] Chiueh, Tzi-cker, "Content-Based Image Indexing," *in Proc. of the 20th VLDB Conf.*, pp.582-593, 1994.

[9] Fan, J. J. and K. Y. Su, "An Efficient Algorithm for Matching Multiple Patterns," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, pp. 339-351, Dec. 1993.

[10] Fan, J. J. and K. Y. Su, "The Design of Efficient Algorithms for Two-Dimensional Pattern Matching," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 7, No. 2, pp. 318-327, Apr. 1995.

[11] Gibbs, S., C. Breiteneder, and D. Tsichritzis, "Audio/Video Database: An Object-Oriented Approach," *in Proc. on Intl. Conf. on Data Eng.*, pp. 381-390, 1993.

[12] Hjelsvold, R. and R. Midstraum, "Modeling and Querying Video Data," in Proc. of the 20th VLDB Conf., pp. 686-694, 1994.

[13] Knuth, D. E., J. H. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Comput.*, Vol 6, pp.323-350, June 1977.

[14] Kuo, C. T., Y. B. Lin, and A. L. P. Chen, "An Approach for Efficient Shot Change Detection on Compressed Video Data," *Proc. of IEEE Intl. Workshop on Multimedia*

Database Management Systems, 1996.

[15] Kuo, C. T. and A. L. P. Chen, "A Content-Based Query Language for Video Databases," *in Proc. of IEEE Multimedia Systems Conf.*, 1996.

[16] Lee, S. Y., M. K. Shan, and W. P. Yang,"Similarity Retrieval of Iconic Image Database," *Pattern Recognition*, Vol. 22, No. 6, pp. 675-682, 1989.

[17] Liu, C. C. and A. L. P. Chen, "Vega: A Multimedia Database System Supporting Content-Based Retrieval," *Journal of Information Science and Engineering*, to appear.

[18] Oomoto, E. and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 5, No. 4, pp. 629-643, Aug. 1993.

[19] Petrakis, E. G. M. and S. C. Orphanoudakis, "Methodology for the Representation, Indexing and Retrieval of Images by Content," *Image and Vision Computing*, Vol. 11, No. 8, pp.504-521, 1993.

[20] Smoliar, S. W. and H. J. Zhang, "Content-Based Video Indexing and Retrieval," *IEEE Multimedia*, Vol. 1, No. 2, pp. 62-72, 1994.

[21] Tonomura, Y. et al., "Structured Video Computing," *IEEE Multimedia*, Vol. 1, No. 3, pp. 34-43, Fall 1994.

[22] Weiss, R., A. Duda, and D. K. Gifford, "Content-Based Access to Algebraic Video," *in Proc. of the Intl. Conf. on Multimedia Computing and Systems*, pp. 140-151, May 1994.

[23] Zhu, R. F. and T. Takaoka, "A Technique for Two-Dimensional Pattern Matching," *Comm. ACM*, Vol. 32, No. 9, pp. 1110-1120, Sep. 1989.

[24] Biliris, A. and E. Panagos, *EOS Users' Guide*, Release 2.2., AT&T Bell Laboratories.

[25] S.Y. Lee, M.K. Shan, and W.P. Yang, "Similarity Retrieval of Iconic Image Database," *Pattern Recognition*, Vol. 22, No. 6, 1992.

[26] N. Dimitrova and F. Golshani, "Motion Recovery for Video Content Classification," *ACM Trans. Information Systems*, Vol. 13, No. 4, pp. 408-439, 1995.

[27] T.T.Y. Wai and A.L.P. Chen, "Retrieving Video Data via Motion Tracks of Content Symbols," *Proc. ACM International Conference on Information and Knowledge Management (CIKM).*

Appendix:

This appendix collects the algorithms for video query processing used in this paper. Algorithms 5, 6, and 7 are used to construct the 1D-List. Given a 1D-String and a 1D-List, Algorithm 8 generates the solution of the 1D-String. Algorithm 9 constructs the 3D-List by applying Algorithm 5 three times. Algorithm 10 refines the 3D-List. Algorithm 11 generates

the solutions of a 3D-String.

Algorithm 5 *Build_1D_List(X, V)*

```
/* input : an extended 1D-String X = I1α1I2α2...αk-1Ik and the OID of a video object */
/* output : a 1D-List L = (X, S, {s_start, s_end}, R, E) */
0:  begin
1:      α0 ← "⇨"
2:      αk ← "⇨"
3:      E ← ϕ
4:      for i = 1 to k
5:      begin
6:          retrieve every symbol object s and Rx(s) where s is in V and Icon(s) = Ii and
7:          store these pairs of symbol object and rank as Si
8:          sort the symbol objects in Si according to their rank value
9:          E ← E ∪ Build_Equ_Relationship(Si, αi-1, αi)
10:     end
11:     R ← ϕ
12:     S0 ← {s_start }
13:     Sk+1 ← {s_end}
14:     for I = 0 to k
15:         R ← R ∪ Build_ST_Relationship(Si, Si+1,αi-1 ,αi,αi+1)
16: end
```

Algorithm 5 describes how to construct a 1D-List for each video object V in the video database with respect to one of the three 1D-Strings of a video query. It first sets the relationship between s_{start} and the first icon and the relationship between s_{end} and the last icon to be "⇨". Then the symbol objects of V which are related to the video query are retrieved from the video index and arranged according to their icon types and rank values (line 6-8). The equivalent relationships (line 9) and the extended spatial-temporal relationships (line 14-15) are built by applying the following two algorithms.

Algorithm 6 *Build_ Equ_Relationship(S, α1, α2)*

```
/* input : a symbol object set S and two extended spatial-temporal */
/*         relationships α1 ,and α2                                */
/* output : an equivalent relationship set E among S               */
0:  begin
1:      E ← ϕ
2:      if (α1 = "⇨") and (α2 = "⇨")
3:          for i = 1 to Num(S) - 1
4:              E ← E ∪ (si, si+1)
5:      else
6:          for i = 1 to Num(S) - 1
7:              if Rx(si) = Rx(si+1)
8:                  E ← E ∪ (si, si+1)
9:              end if
10:     end if
```

```
11:     return E
12: end
```

Algorithm 6 constructs equivalent relationships among a symbol object set S with respect to two extended spatial-temporal relationships α_1 and α_2. As we have discussed in Definition 15, if both α_1 and α_2 are precedent relationships (line 2), S forms a single equivalent group. Since the symbol objects in S are sorted and equivalent relationships possess transitivity property, we just link each pair of adjacent symbol objects in S (line 3-4). If one of α_1 and α_2 is not precedent relationships, the symbol objects which have the same rank values in S will form an equivalent group (line 7-8). The time complexity of this algorithm is O(n) where n is the number of symbol objects in the equivalence group S.

Algorithm 7 *Build_ ST_Relationship(S_i, S_{i+1},α_{i-1} ,α_i,α_{i+1})*

```
/* input : two symbol object set S_i and S_{i+1} and three extended spatial-temporal */
/*         relationships α_{i-1} ,α_i, and α_{i+1}                                    */
/* output : an ST relationship set R                                                  */
0:  begin
1:      R ← φ
2:      if i = 0                          /* build ST relationships from s_start to S_1 */
3:      begin
4:          R ← R ∪ (s_start, s_{1,1})
5:          for j = 1 to Num(S_1) - 1
6:              if there do not exist an equivalent relationship between s_{1,j} and s_{1,j+1}
7:                  R ← R ∪ (s_start, s_{1,j+1})
8:              endif
9:      end
10:     else if i = k                     /* build ST relationships from S_k to s_end */
11:     begin
12:         R ← R ∪ (s_{1,Num(Sk)}, s_end)
13:         for j = 1 to Num(S_k) - 1
14:             if there do not exist an equivalent relationship between s_{k,j} and s_{k,j+1}
15:                 R ← R ∪ (s_{k,j}, s_end)
16:             endif
17:     end
18:     else    /* build ST relationships from S_i to S_{i+1}                         */
19:             /* Assume S_i = <G_1, G_2, ..., G_m> and S_{i+1} = <G_1', G_2', ..., G_n'> */
20:             /* where each Gj and Gj' is an equivalent group                       */
21:     begin case
22:     case (α_{i-1} ="⇨") and (α_i ="⇨") and (α_{i+1} ="⇨")
23:         for each s_i in G_1
24:             for each s_j in G_1'
25:                 if (R_x(s_i) < R_x(s_j)) and (R_x(s_{i+1}) ≥ R_x(s_j)) and (R_x(s_i) ≥ R_x(s_{j-1}))
26:                     R ← R ∪ (s_i, s_j)
27:                 end if
28:     case (α_{i-1} = "⇨") and (α_i = "⇨") and ((α_{i+1} = "≡") or (α_{i+1} = "|_n"))
```

```
29:         for each s_i in G_1
30:             for each first symbol object s_j in G_j', 1≤ j ≤ n
31:                 if (R_x(s_i) < R_x(s_j)) and (R_x(s_{i+1}) ≥ R_x(s_j))
32:                     R ← R ∪ (s_i, s_j)
33:                 end if
34:     case ((α_{i-1} = "≡") or (α_{i-1} = "|_n")) and (α_i = "⇨") and (α_{i+1} = "⇨")
35:         for each last symbol object s_i in G_i, 1≤ i ≤ m
36:             for each s_j in G_1'
37:                 if (R_x(s_i) < R_x(s_j)) and (R_x(s_i) ≥ R_x(s_{j+1}))
38:                     R ← R ∪ (s_i, s_j)
39:                 end if
40:     case ((α_{i-1} = "≡") or (α_{i-1} = "|_n")) and (α_i = "⇨") and ((α_{i+1} = "≡") or (α_{i+1} = "|_n"))
41:         for each last symbol object s_i in G_i, 1≤ i ≤ m
42:             for each first symbol object s_j in G_j', 1≤ j ≤ n
43:                 if (R_x(s_i) < R_x(s_j))
44:                     R ← R ∪ (s_i, s_j)
45:                 end if
46:     case (α_i = "≡")
47:         for each last symbol object s_i in G_i, 1≤ i ≤ m
48:             for each first symbol object s_j in G_j', 1≤ j ≤ n
49:                 if (R_x(s_i) = R_x(s_j))
50:                     R ← R ∪ (s_i, s_j)
51:                 end if
52:     case (α_i = "|_k")
53:         for each last symbol object s_i in G_i, 1≤ i ≤ m
54:             for each first symbol object s_j in G_j', 1≤ j ≤ n
55:                 if (R_x(s_i) - R_x(s_j) = k)
56:                     R ← R ∪ (s_i, s_j)
57:                 end if
58:     end case
59:     end if
60:     return R
61: end
```

Algorithm 7 constructs the extended spatial-temporal relationships between the symbol objects of the two symbol object sets S_i and S_{i+1} according to the three extended spatial-temporal relationships α_{i-1} ,α_i, and α_{i+1}. Its time complexity is O(n) where n is the total number of symbol objects in the equvalence groups S_i and S_{i+1}. The time complexity of Algorithm 5 is O(k·nlogn), where k is the number of icons in the video query and n is the maximal number of the symbol objects belonging to each icon.

Algorithm 8 *Generate_1D_Result(L)*

```
/* input : a 1D-List L = (X, S, {s_start, s_end}, R, E)                    */
/* output : the 1D-List L' which represents the one-dimensional result */
0:  begin
1:  /* Forward Remove */
```

```
2:      Stack ← ϕ
3:      Push(Stack, s_start)
4:      for each s′ such that (s, s′) ∈ R
5:          Push(Stack, s′)
6:      while (Stack ≠ ϕ) do
7:      begin
8:          Pop(Stack, s)
9:          if exist (s′, s) ∈ E
10:             remove (s′, s) from E
11:         end if
12:         if exist (s, s′) ∈ E
13:             Push(Stack, s′)
14:         end if
15:         if exist (s, s′) ∈ R
16:             if s′ = s_end
17:                 Pop(Stack, s) until Icon(s) = I_1
18:                 Push(Stack, s)
19:             else
20:                 Push(Stack, s′)
21:             end if
22:         end if
23:     end
24: /* Backward Remove */
25:     Stack ← ϕ
26:     Push(Stack, s_end)
27:     for each s′ such that (s′, s) ∈ R
28:         Push(Stack, s′)
29:     while (Stack ≠ ϕ) do
30:     begin
31:         Pop(Stack, s)
32:         if exist (s, s′) ∈ E
33:             remove (s, s′) from E
34:         end if
35:         if exist (s′, s) ∈ E
36:             Push(Stack, s′)
37:         end if
38:         if exist (s′, s) ∈ R
39:             if s′ = s_start
40:                 Pop(Stack, s) until Icon(s) = I_k
41:                 Push(Stack, s)
42:             else
43:                 Push(Stack, s′)
44:             end if
45:         end if
46:     end
47: end
```

Algorithm 9 *Build_3D_List((X, Y, T), V)*

```
/* input : an extended 3D-String (X, Y, T) and the OID of a video object */
/* output : a 3D-List L = (L_x, L_y, L_t) */
0:  begin
1:      L_x = Build_1D_List(X, V)
2:      L_y = Build_1D_List(Y, V)
3:      L_t = Build_1D_List(T, V)
4:      return (L_x, L_y, L_t)
5:  end
```

Algorithm 9 constructs the 3D-List by applying Algorithm 5 three times. Its time complexity is the same as Algorithm 5.

Algorithm 10 *Refine_3D_Result(L)*

```
/* input : a 3D-List L = (L_x, L_y, L_t) where L_x = (X, S_x, {s_start, s_end}, R_x, E_x),   */
/*         L_y =(Y, S_y, {s_start, s_end}, R_y, E_y), L_t =(T, S_t, {s_start, s_end}, R_t, E_t)   */
/* output : a refined 3D-List L' = (L_x', L_y', L_t')                                        */
0:  begin
1:      do
2:          S_old ← S_x ∪ S_y ∪ S_t
3:          S_x ← S_old
4:          S_y ← S_old
5:          S_y ← S_old
6:          L_x' ← (X, S_x, {s_start, s_end}, R_x, E_x)
7:          L_y' ← (Y, S_y, {s_start, s_end}, R_y, E_y)
8:          L_t' ← (T, S_t, {s_start, s_end}, R_t, E_t)
9:          Generate_1D_Result(L_x')
10:         Generate_1D_Result(L_x')
11:         Generate_1D_Result(L_x')
12:         S_new ← S_x ∪ S_y ∪ S_t
13:     until S_new = S_old
14: end
```

Algorithm 11 *Generate_Complete_Sol(L)*

```
/* input : a 3D-List L = (L_x, L_y, L_t) where L_x = (X, S_x, {s_start, s_end}, R_x, E_x),   */
/*         L_y =(Y, S_y, {s_start, s_end}, R_y, E_y), L_t =(T, S_t, {s_start, s_end}, R_t, E_t)   */
/* output : the complete solution sets of the 3D-String(X, Y, T)                             */
0:  begin
1:      for each path s_start α0 s1 α1 s2 α2 ... α_{p-1} s_p α_p s_end in Lx
2:          for each path s_start β0 s1' β1 s2' β2 ... β_{q-1} s_q' β_q s_end in Ly
3:              for each path s_start γ0 s1'' γ1 s2'' γ2 ... γ_{r-1} s_r'' γ_r s_end in Lt
4:              begin
5:                  S ← { s1, s2, ..., sp} ∩ { s1', s2', ...,sq'} ∩ { s1'', s2'', ...,sr''}
6:                  if NumOfIcon(S) = k
7:                      output(S)
8:                  end if
9:              end
10: end
```