Computer Vision and Image Understanding
Vol. 75, Nos. 1/2, July/August, pp. 25–45, 1999
Article ID cviu.1999.0764, available online at http://www.idealibrary.com on IDEAL®

A Distance Measure for Video Sequences*

Donald A. Adjeroh,† M. C. Lee, and Irwin King

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
E-mail: donald@cse.cuhk.edu.hk, mclee@cse.cuhk.edu.hk, king@cse.cuhk.edu.hk

Video is a unique multimedia data type, in that it comes with distinguished spatio-temporal constraints. Content-based video retrieval thus requires methods for video sequence-to-sequence matching, incorporating the temporal ordering inherent in a video sequence, without losing sight of the visual nature of the information in the sequence. Such methods will require reliable measures of similarity between the video sequences. In this paper, we formulate the problem of video sequence-to-sequence matching as a pattern-matching problem and propose the vstring edit distance as a suitable distance measure for video sequences. © 1999 Academic Press

1. INTRODUCTION

In general, content-based video retrieval is concerned with the indexing and retrieval of video information based on the actual contents of the video sequence. The first stage of this process is the initial partitioning (segmentation) of the video data into its constituent scenes and the identification of the various editing effects in the video [3, 18]. The next stage involves the analysis of the individual scenes to provide further information for fine-grained access to the digital video. One such analysis results in the generation or selection of representations for each video scene. A typical representation used here is the video key frame. Others are compact representations, such as video mosaics [27], layered representations [24], and super resolution frames [25]. Further stages of the retrieval process could involve the classification or clustering of the partitioned video, based on certain characteristics of the scenes, such as motion complexity or the activity in the scene [2, 12]. From these, it could then become possible to address the more difficult problem of semantic access to the video content [12].

So far, however, the problem of matching video sequences for possible similarity has attracted little attention. The current methods are generally based on the use of the above representations (such as the key frame) as an image, and then using image matching methods to compare the scenes [13, 23]. The problem with this simple image-based approach is the loss of the temporal information inherent in a video sequence. Even when the key frames or surrogates have been used to identify probable scenes that may be similar to an input query, a human observer is still required to manually verify if the returned sequences are actually similar to the query sequence. Considering the large number of sequences in typical video databases, effective and efficient methods are still required to perform such verifications automatically. Such methods will certainly call for reliable measures of the similarity or difference between video sequences.

The spatio-temporal constraints that accompany video data types are one of the unique characteristics of video information. The importance of the temporal constraints has led to recent efforts to incorporate them in video retrieval. The question is how to capture such important video characteristics for use in content-based retrieval. For instance, motion cues between adjacent frames have been proposed for video retrieval [12, 19, 34]. Others have tried to model the temporal information directly by treating the video data in its natural form—as an ordered sequence of frames [1, 7, 33]. This leads directly to video sequence similarity matching, whereby video is treated as a temporally ordered sequence, rather than as a mere collection of images. Video sequence matching is primarily motivated by the practical need to incorporate the inherent temporal constraints in video sequences in content-based retrieval. Results from such an endeavor will be equally of interest in other application areas, such as music and audio retrieval, medical imaging [28], biomedical monitoring [32], crime investigation (copyright infringement), and TV broadcasting. With the new popularity of applications involving content-based access to digital video, video sequence matching stands to play a more important role in various areas in modern-day computing.

The method used to address the video sequence similarity-matching problem generally depends on the technique used to represent the sequences. For instance, if the video information in a given sequence is treated as a time series signal, then ideas from time series analysis can be used to compare two such signals. Chang and Lee [9] used the bounding box approach to model transitions between frames and defined some similarity measures based on the bounding box representation. Another method with which the information in a video sequence can be captured is the string representation. Here, the video sequence as described by a sequence of feature values is transformed into

* This work was supported in part by RGC Grant CUHK H164/97E, UGC Direct Grant (Project 1D 2050192), RGC Hong Kong CUHK 4176/97E and ISF Hong Kong AF/17/95.

† Corresponding author.

1077-3142/99 $30.00

a sequence of symbols. For a given feature, this will involve the initial transformation of the real-valued feature values into some discrete classes. Each symbol in the string will represent a class, and the set of all symbols will form the alphabet. Thus, the total number of symbols (the alphabet size) will depend on the number of classes used. We call such a representation of the video sequence a *video string*, or *vstring* for short. Yazdani and Ozsoyoglu [33] defined some statistics based on the lengths of image sequences and used them for image sequence matching. Adjeroh, King, and Lee [1] described the general problems in video sequence similarity matching and proposed the *vstring* as an appropriate representation for the video sequence when the objective is similarity matching. In this paper, we propose a similarity measure for video sequences based on the vstring representation.

Syntactic/structural pattern recognition is an area that was pioneered by Fu [6, 14, 15] and has found applications in texture analysis, shape recognition [29], and image retrieval [9, 8]. Extensions into the 3D and higher dimensions have also been proposed [8, 14]. In general, the main objective is to model the physical structure of the objects in an image. Though these also lead to a string representation, the vstring representation is different in terms of the basic primitives from which it is generated. Rather than the physical objects, the index feature values form the basis for the vstring representation. The vstring models the basic transitions *between* image sequences, with no special emphasis on the spatial or structural relationship between objects *within* the images. Like in ordinary strings, the vstring is a sequence of symbols. The symbols in the vstring are, however, generally multivalued, rather than the simple presence/absence symbols encountered in other strings. Moreover, to account for the various visual cues with which the video information can be analyzed, vstrings are typically multidimensional. Further, to model the special functions involved in digital video, new edit operations are required for the vstring. One advantage of the vstring representation is that, with appropriate modifications, the problem of video sequence similarity matching in multimedia information retrieval can be turned into the more familiar problem of approximate string matching. Techniques for approximate string matching can then be used for video similarity comparison. However, the major problem of computational complexity for traditional approximate string matching will also have to be addressed for the video string representation. Fast methods for general pattern matching is an area that has long been investigated, especially for exact pattern matching [20]. Equivalently, fast algorithms have been proposed for approximate string matching [21, 22, 30]. A general review of fast pattern matching algorithms is presented in [10].

In this paper, we focus on devising a suitable similarity measure for video sequences. The next section describes the general problem of video sequence matching, the vstring representation, and the concept of string pattern matching. In Section 3, the vstring edit distance is proposed as a distance measure for video sequences. Section 4 presents an alternative method to compute the vstring distance. Normalizations for the vstring distance are described in Section 5. Experimental results are presented in Section 6.

2. VIDEO SEQUENCE MATCHING AND STRING PATTERN MATCHING

DEFINITION 1. A *video sequence* is a set of temporally ordered scenes. A *scene* (or *shot*) is a set of frames between two adjacent scene breaks, as may be caused by camera operations (such as a cut), or by special effect operations—such as a fade or dissolve. The number of frames in a scene depends on the specific contents of the scene. A video *frame* is simply a single image. A frame can be divided into different subparts. When a frame is so divided, we use the term *subframe* to denote any of the subparts. We assume that whenever a frame is divided, the subframes are all equal in area.

Given a query video sequence and a database of video sequences, the *video sequence-searching problem* is to find one or all the occurrences of the query video sequence in the database. The problem then is to search the entire video database for the requested query video sequence, producing a list of the positions in the database sequence where a match starts (or ends). When the problem is exact matching of the sequences, it is not difficult to compute some simple statistics with which the matching can be performed. Exact matching is, however, not very suitable in multimedia information systems, especially those involving visual data. A more useful variant of the problem is that of video sequence *similarity searching* which will in turn depend on the methods for video sequence *similarity matching.*

2.1. The Problem

Since we are often more interested in similarity (rather than exact) matching, there is need for some measure to indicate the degree of similarity between two sequences. Let given two video sequences A and B be represented by their p-feature values at each of the temporal indices t and r,

$$\begin{aligned} A &= [a_1(t), a_2(t), \ldots, a_p(t)]^{\mathrm{T}}, \\ B &= [b_1(r), b_2(r), \ldots, b_p(r)]^{\mathrm{T}}, \end{aligned} \tag{1}$$

where $r = 0, 1, 2, \ldots, n$; $t = 0, 1, 2, \ldots, m$, x^{T} stands for transpose of x. Given some possible variations in the p features from A and/or B, the problem is to find a mapping $f: A \times B \times \theta \rightarrow \Re$, such that f is independent of n, m the sequence lengths and still robust under the variations, as captured by θ. We should be able to normalize the resulting values of f to some given ranges, for instance [0 1], such that the degree of similarity increases uniformly from the minimum value (perfect mismatch) to the maximum value (perfect match). Different measures can be used to achieve the required mapping. The measure could be a distance function or a similarity function and need not necessarily be a metric. Whether the two sequences are similar (matches) or not

will then depend on some predefined threshold of similarity. The threshold can be chosen based on the application, or it may be stipulated by the user.

Conceptually, A can represent the entire database, while B is just a short query sequence. Here, we would be interested in knowing if there exists any subsequence of A that is similar to B. The problem of exact video sequence matching is thus easily handled by an appropriate choice of the threshold. Three basic types of matching can be identified in video sequence matching:

(i) *scene-to-scene matching*: check if two scenes are similar;
(ii) *scene-to-sequence matching*: check if a scene similar to the query scene occurs in the database sequence;
(iii) *sequence-to-sequence matching*: check if a sequence similar to a query sequence occurs in the database sequence. The query can contain more than one scene.

The case of sequence-to-scene matching is handled by simply assuming that the database sequence is the longer of the two sequences. This has no effect on the actual matching process. We observe that (ii) is a generalization of (i) and will make use of the methods for (i). Similarly, (iii) is a generalization of (ii) and its solution will depend on the solutions to (ii). The basic problem, thus, is finding solutions to (i): the scene-to-scene matching problem.

2.2. The *vString* Representation

The vstring describes the sequence of feature values from the video as a sequence of symbols. Each symbol in the string represents a class and the set of all symbols form the alphabet. With a nonuniform distribution of the feature values, an equiprobable classification will require knowledge of the probability distribution of the feature values. If we assume an equiprobable distribution for the feature values (i.e., all the symbols in the alphabet are equally probable), a simple method that can be used to transform the features into vstrings is the simple uniform quantization.

Let Σ be the symbol alphabet, f_v be a feature value, and $\max f_v$ and $\min f_v$ be the respective maximum and minimum values for a given index feature. The quantization step size is given by

$$\Delta = \frac{\max f_v - \min f_v}{|\Sigma|}. \tag{2}$$

The quantization level to which a given f_v belongs is then obtained using:

$$q(f_v) = i \quad \text{if } (i-1)\cdot\Delta \le f_v < i\cdot\Delta;\ i = 1, 2, \ldots, |\Sigma|. \tag{3}$$

If a feature value belongs to the ith quantization level, we assign the ith symbol to it. For multiple features, we may have different classifications, leading to multiple alphabets, with possibly different cardinality. In such a case, we will have multidimensional video strings, with the strings from each feature forming a dimension. Usually, symbols in traditional text strings are taken as presence/absence symbols—that is, the symbols either appear or do not appear in the string, and two different symbols are assumed not to have much else in common. For the vstring, when the symbols are taken from an alphabet obtained from the classification of real-valued features, it is safe to assume that nearby classes are related. That is, a feature value that belongs to the first class is nearer to another feature value that belongs to the second class than to one that belongs to the last class (assuming more than two classes). This implies that the symbols in the video string will be multivalued. This modification is needed to improve the accuracy of the similarity measurement using vstrings and will have some important implications in defining distances between video strings. For some other types of classification (e.g., semantic classification of the sequences) the symbols can be treated as the traditional presence/absence symbols.

The advantage of the vstring representation is that it is fairly general—we can represent different types of video scene classifications using string sequences. For instance, we can easily classify, based on some semantic descriptions or using quantitative features from the video [1], motion vectors, angles, color, etc. More importantly, the vstring representation provides an intuitive method to model various characteristics and phenomena observed in a video sequence—such as repetitions, reverse, fast forward, and video scene breaks. Example 1 taken from [1] shows how the basic video scene transitions such as fast forward, slow motion, reverse, and partial reverse can be modeled by the vstring representation.

EXAMPLE 1. (sVstring representation for different video transitions). The symbols a, b, c, etc. represent the different classes to which the feature values are grouped. A video frame is represented by a symbol. The special marker *$* stands for video scene break:

Original sequence: *aaabbbcccdddeee$vvvxxxyyy*;

Fast forward:
skip = 1: speed = ×2: *aabccdee$vxxy*
skip = 2: speed = ×3: *abcde$vxy*
skip = 3: speed = ×4: *abce$vy*

Slow motion:
speed = $\frac{1}{2}$ original speed: *aaa aaa bbb bbb ccc ccc ddd ddd eee eee$vvv vvv xxx xxx yyy yyy*
speed = $\frac{1}{4}$ original speed: *aaa aaa aaa aaa bbb bbb bbb bbb . . . eee eee eee eee$vvv vvv vvv vvv . . .*

Reverse: *yyy xxx vvv$eee ddd ccc bbb aaa*

Partial reverse (due to video editing):
Case 1: *bbbccc* and *dddeee* transposed: *aaa dddeee bbbccc $ vvvxxxyyy*
Case 2: *bbb* and *vvv* transposed: *aaa vvvcccdddee e$ bbb $ xxxyyy*

Note. Spaces between symbols are inserted only for clarity. For partial reverse, the first, second, and fourth scene breaks

in Case 2 are required since the edit operation involved two different scenes. In Case 1, the editing involved only frames in the same scene. Fast reverse is very similar to fast forward.

2.3. String Pattern Matching and Edit Distances

Given a database string A and a query string (the pattern) B, the string pattern matching problem is to find the first occurence (or all occurences) of B in A. Approximate pattern matching is a variant of the pattern matching problem in which k-symbol differences can be allowed in the match. That is, a symbol can be in A but not in B, or a symbol can be in B but not in A, and A and B can differ in certain positions, but the number of positions where they differ should not be more than k. The distance between two strings is traditionally calculated using the string edit distance.

DEFINITION 2. Given two strings $A: a_1a_2 \cdots a_n$ and $B: b_1b_2 \cdots b_m$, over an alphabet Σ, a set of allowed edit operations, and a unit cost for each operation, the *edit distance* is the minimum number of edit operations required to transform one string into the other.

2.3.1. Edit Operation

DEFINITION 3. An *edit operation*, usually written as $(x \to y)$, is a pair $(x, y) \neq (\varepsilon, \varepsilon)$, of strings where $|x| \leq 1$ and $|y| \leq 1$ (ε represents the zero-length empty symbol). When we apply the edit operation $(x \to y)$ on an input string S_I to obtain an output string S_O, we say the input string S_I is transformed into the output string S_O, via the edit operation $(x \to y)$. Or simply that x is transformed into y. That is, there exist some strings S_1 and S_2, such that $S_I = S_1xS_2$ and $S_O = S_1yS_2$.

With the above definition, x and y are constrained to be single symbols. Three basic types of edit operations are used: *ins*—insertion of a symbol, $(\varepsilon \to a)$; *del*—deletion of a symbol, $(a \to \varepsilon)$; and *subs*—substitution of one symbol for another $(a \to b)$. To any given edit operation $(x \to y)$, a cost $c(x \to y)$ is assigned. The value of the cost is determined by use of a weighting function.

2.3.2. Edit Sequences and Edit Distance

DEFINITION 4. An *edit sequence* (or edit path) is an ordered set of edit operations that transforms one string into another.

To transform a string A into another string B, one will typically need to apply different edit operations: $S_E = s_1s_2 \cdots s_l$, where $s_i \in \{ins, del, subs\}$. The cost of a given edit sequence is determined by the cost of the individual edit operations that make up the sequence:

$$c(S_E) = \sum_{i=1}^{l} c(s_i), \quad s_i \in \{ins, del, subs\}. \tag{4}$$

The cost is independent of the order in which the constituent edit operations are applied. Given two strings, A and B, there may be more than one edit sequence that transforms string A into string B. Let $S_{A \to B}$ represent the set of all edit sequences that transform A into B. The edit distance $D(A, B)$ is determined by using the edit sequence with the minimum cost, that is,

$$D(A, B) = \min\{c(S_E) \mid S_E \in S_{A \to B}\}. \tag{5}$$

We can constrain the cost for each edit operation $c(x \to y)$ to be a distance metric by ensuring the following conditions are always met:

(i) $c(x \to y) \geq 0$ (nonnegative);
(ii) $c(x \to x) = 0$ (reflectivity);
(iii) $c(x \to y) = c(y \to x)$ (symmetry);
(iv) $c(x \to y) \leq c(x \to z) + c(z \to y)$ (triangular inequality).

With the above constraints and since the edit distance will always select the path with the minimum cost, it is easy to prove the following lemma.

LEMMA 1. *If $c(x \to y)$ is a metric, the edit distance $D(A, B)$ between strings A and B is also a metric. That is, $D(A, B) \geq 0$; $D(A, A) = 0$; $D(A, B) = D(B, A)$; $D(A, B) \leq D(A, C) + D(C, B)$.*

We note that the requirement for a metric is only for convenience, but not a necessity. For instance, with symmetric costs, we will not need to worry about which string is used as the query string or the database string. On the other hand, the requirement for triangular inequality is often not met in most multimedia retrieval environments. The edit distance between $A: a_1a_2 \cdots a_n$ and $B: b_1b_2 \cdots b_m$ is usually determined by use of some recurrence relations [26, 31]:

initializations:

$$\begin{aligned} D_{0,0} &= 0, \\ D_{i,0} &= D_{i-1,0} + \alpha_{del}(a_i), \\ D_{0,j} &= D_{0,j-1} + \alpha_{ins}(b_j); \end{aligned}$$

main recurrence:

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + \alpha_{del}(a_i) & \text{(deletion)}, \\ D_{i-1,j-1} + \alpha_{subs}(a_i, b_j) & \text{(substitution)}, \\ D_{i,j-1} + \alpha_{ins}(b_j) & \text{(insertion)}, \end{cases}$$

where $1 \leq i \leq |A| = n$; $1 \leq j \leq |B| = m$; α_{del}, α_{ins}, and α_{subs} are the respective cost of deletion, insertion, and substitution edit operations. Example 2 below shows the edit distance between two strings, for two different cost functions, α.

EXAMPLE 2. Edit distance between two sets of strings using two different cost functions: $\alpha = [\alpha_{del} \ \alpha_{ins} \ \alpha_{subs}] = [1\ 1\ 1]$; and $\alpha = [1\ \ 1\ \ 2]$. Table 1 shows the computation procedure. The highlighted path (underlined and bold) represents one of the possible minimum cost edit sequences. For (a), the path corresponds

TABLE 1
Edit Distance between Two Sets of Strings and for Two Different Cost Functions

$A = [1, 2, 1, 2, 2, 2]$, $B = [2, 1, 2, 1, 1, 1]$;
$\alpha = [1\ \ 1\ \ 1]$; $D(A, B) = 4$

					B			
	D	ε	2	1	2	1	1	1
	ε	**0**	1	2	3	4	5	6
	1	**1**	1	1	2	3	4	5
	2	2	**1**	2	1	2	3	4
A	1	3	2	**1**	2	1	2	3
	2	4	3	2	**1**	2	2	3
	2	5	4	3	2	**2**	3	3
	2	6	5	4	3	3	**3**	**4**

(a)

$A = [1, 2, 1, 2, 2, 2]$, $B = [2, 1, 2, 1, 1, 1]$;
$\alpha = [1\ \ 1\ \ 2]$; $D(A, B) = 6$

					B			
	D	ε	2	1	2	1	1	1
	ε	**0**	1	2	3	4	5	6
	1	**1**	2	1	2	3	4	5
	2	2	**1**	2	1	2	3	4
A	1	3	2	**1**	2	1	2	3
	2	4	3	2	**1**	2	3	4
	2	5	4	3	**2**	3	4	5
	2	6	5	4	**3**	**4**	**5**	**6**

(b)

$A = [2, 3, 1, 2]$, $B = [1, 2, 3, 1, 3, 1, 3]$;
$\alpha = [1\ \ 1\ \ 1]$; $D(A, B) = 4$

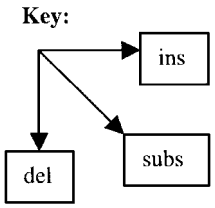
						B			
	D	ε	1	2	3	1	3	1	3
	ε	**0**	**1**	2	3	4	5	6	7
	2	1	1	**1**	2	3	4	5	6
A	3	2	2	2	**1**	2	3	4	5
	1	3	2	3	2	**1**	2	3	4
	2	4	3	2	3	2	**2**	**3**	**4**

(c)

$A = [2, 3, 1, 2]$, $B = [1, 2, 3, 1, 3, 1, 3]$;
$\alpha = [1\ \ 1\ \ 2]$; $D(A, B) = 5$

						B			
	D	ε	1	2	3	1	3	1	3
	ε	**0**	**1**	2	3	4	5	6	7
	2	1	2	**1**	2	3	4	5	6
A	3	2	3	2	**1**	2	3	4	5
	1	3	2	3	2	**1**	2	3	4
	2	4	3	2	3	**2**	**3**	**4**	**5**

(d)



to the alignment:

$$|1|2|1|2|2|2|\ \ |$$
$$|\ \ |2|1|2|1|1|1|$$

This represents 1 deletion, 2 substitutions, and 1 insertion, corresponding to the following edit operations: delete(1) in A, substitute(2, 1), substitute(2, 1), insert(1) in A. The highlighted edit path in (c) corresponds to the alignment:

$$|\ \ |2|3|1|2|\ \ |\ \ |$$
$$|1|2|3|1|3|1|3|$$

This corresponds to the operations: insert(1) in A, substitute (2, 3), insert(1) in A, insert(3) in A.

The above recurrence implies that an $O(mn)$ time is required for the edit distance. Some other measures used to evaluate the similarity between strings include the longest common subsequence (LCS), counting, and scoring functions.

3. THE VIDEO STRING EDIT DISTANCE

Motivated by the traditional methods used in the video editing process—assemble and insert editing [11], and the edit distance used in string pattern matching, we propose a method for matching video sequences. The technique is suitable for both frame-by-frame sequence representation and comparison and for shot-by-shot comparisons. We call the resulting similarity indicator the *vstring edit distance*.

The *vstring* edit distance is based on an intuitive idea. Given two video sequences (now represented by their vstrings), we assume that at some initial state the two sequences were the same (with no difference) and that the current state of the sequences is a result of zero or more video editing operations. Here, by video editing, we refer to the process of arranging individual frames,

shots, or sequences into an appropriate order [11]. In addition to the possible temporal rearrangements, our notion of video editing also includes possible changes in the actual *content* of the frames making up the shots. The string edit distance has been used in various applications, such as spell checking and DNA sequence matching. When the problem is content-based access to digital video using the vstrings, the traditional string edit distance is, however, inadequate to capture the similarity between the vstrings. The limitations can be illustrated with a simple hypothetical example.

EXAMPLE 3. Assume we have three sequences all of the same length. Assume further that each has been transformed into its respective vstring: v1, 11111; v2, 33333; v3, 88888, using an alphabet size of 8: $\Sigma = \{1, 2, 3, \ldots, 8\}$. Practically, this will correspond to an all-black frame sequence, frame sequences with some shades of grey; and an all-white frame sequence. With the traditional string edit distance, all the sequences will be equally distant (an edit distance of 5, using a unit cost for each edit operation). In terms of visual content, however, v1 will be observed as being much closer to v2 than to v3. The traditional string edit distance as described previously cannot capture such content-dependent details.

It is then obvious that traditional string edit distances will be inadequate to cope with video strings. First, the above content requirement implies that the symbols in the video string will have some meaning (rather than traditional presence/absence symbols used in text or DNA strings). This further implies that video string edit distances will have to contend with the values attached to the symbols in the alphabet. The value typically depends on the classification method adopted and the particular features from which the vstring is derived. Second, the three basic edit operations (*ins, del, subs*) will not be adequate in modeling certain characteristics found in the vstring, such as repetitions. Further, the cost of a video string edit operation will typically be affected by the parameters of the edit operation. For instance, the cost of some operations will depend on the number of symbols in their input and output strings. We therefore need to make appropriate considerations with respect to the special nature of vstrings, the unique characteristics of video sequences, and the different types of transitions that may occur in such sequences.

3.1. vString Edit Operations

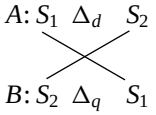
Video sequences often involve special video edit operations (such as those used to produce special effect transitions), special video functionalities (such as fast forward/reverse), or some form of frame skipping (used to improve the speed of processing). To account for the differences that may arise from these unique aspects of video, new edit operations are required in computing the edit distance between two video sequences. For the vstring distance, we need to make a slight modification to the definition of an edit operation. We relax the constraint ($|x| \leq 1$, $|y| \leq 1$) on the length of the strings involved in the edit operation.

DEFINITION 5. The *vString edit operation*, written as $(x \rightarrow y)$, is a pair $(x, y) \neq (\varepsilon, \varepsilon)$ of strings, where $|x| \geq 0$ and $|y| \geq 0$. That is, rather than mere single symbols, x, y (the input and output of the operation) there could be strings with more than one symbol.

This modification becomes important when we consider some new edit operations required for video strings, especially those that act on blocks of symbols, such as the *block-swap* or *fusion* operation. For the vstring, we extend the traditional edit distance by defining some new edit operations, namely *swap/transposition, break, and fusion/fission* operations. Generally, for a given alphabet Σ, the vstring edit operation O_p can be represented as $O_p = \{{}^{a}_{b}O\}$; i.e., *insertion*: $a = \varepsilon, b \in \Sigma$; *deletion*: $a \in \Sigma, b = \varepsilon$; *substitution*: $a, b \in \Sigma$; *swap*: $a, b \in \Sigma^*$; *fusion*: $a, b \in \Sigma^*$; *break*: $a, b \in \aleph$, where $\aleph = \{\varepsilon, \$\}$, $\aleph \cap \Sigma = \emptyset$, Σ is the vstring alphabet, and Σ^* stands for any combination of symbols in Σ. We can then use the new edit operations and adopt an approach similar to that used for traditional edit distances to define a corresponding distance measure between video strings. The three new edit operations are described below.

SWAP. **Interchange two symbols (or blocks of symbol) in one of the strings:** $abcde \rightarrow adcbe$, $abdc \rightarrow adbc$ **(transpose** *b* **and** *d)*; $abdc \rightarrow cbda$ **(transpose** *a* **and** *c)*. We note that in video, apart from the temporal positioning, the actual contents of the frames in the sequence are also important in assessing the similarity between sequences. Thus, the transposition operation will be useful in handling the special transitions such as a partial reverse, primarily caused by video editing (see Example 1). Though the temporal ordering may not be the same in these cases, the contents of the scene may still be viewed as similar by the human observer, since the frames basically contain the same information.

The swap/transposition edit operation can be characterized by the size of the strings to be swapped, the number of symbols separating them in the database string, and the number of symbols separating them in the query string. This is illustrated by



S_1 and S_2 are the strings (not necessarily single symbols, i.e. $|S_1| \geq 1$, $|S_2| \geq 1$) to be swapped, Δ_d is the number of symbols separating them in the database string, and Δ_q is the corresponding size of the separation in the query string. Depending on the values of Δ_d and Δ_q, we can define three variants of the swap edit operation:

Δ-*swap*: $\Delta_d = \Delta_q = \Delta$; the swap operation can be applied to any of the sequences (the database or the query) at the same cost. This is also called a *transposition* operation.

Δ_d-*swap*: $\Delta_d < \Delta_q$; the swap operation will be applied to the database string.

Δ_q-*swap*: $\Delta_q < \Delta_d$; the swap operation will be applied to the query string.

In all cases, swapping is performed on *either* the database or the query string (but not both), depending on the one that will result in the minimum cost. The formulation above is fairly general. For instance, in [21] $|S_1| = |S_2| = 1$ in all cases, and transposition is only valid for adjacent symbols ($\Delta_d = \Delta_q = 1$). With the above formulation, however, we can introduce a new edit operation needed for video strings—the *block-swap* operation. Unlike the usual swap or transposition edit operation, here we can swap blocks of symbols in one single operation, rather than through a repeated use of the swap operation. Appendix A shows how to compute the cost for the swap edit operation.

FUSION/FISSION. Fusion merges a consecutive stream of the same symbol into a single symbol: $aaa \rightarrow a$; fission converts a single symbol into a stream of symbols all of the same type: $a \rightarrow aaa$. This is needed to deal with the repetitive nature of symbols in a video string. A single symbol can be split into many symbols of the same type (*fission*). Similarly, consecutive symbols of the same type can be merged into a single symbol (*fusion*).

Let $[aa \cdots a]$ (p symbols) be represented as a^p. The fusion/fission operation then performs a simple transformation: $a^p \rightarrow a^t$, $t = f(p) = 1, 2, \ldots$. Thus, $f(p)$ determines the extent of the fusion/fision operation. For example, if $f(p) = |\frac{p}{r}|$, with $r = 3$, the following will result from the application of the fusion operator: $aa \rightarrow a$; $aaa \rightarrow a$; $aaaa \rightarrow aa$; $a^6 \rightarrow aa$; $a^7 \rightarrow a^3 = aaa$. The choice of the parameter r can be made based on the application, or based on the length of the database and query sequences. Typically, $t \leq \lceil n/m \rceil$, since in the extreme case, the pattern (query string) will be made up of m identical symbols; i.e., $B = b_1 b_2 \cdots b_m = b^m$. The fusion/fission operation is, thus, a form of normalization or scaling[1] on the original strings. It also provides a natural way for handling special video frame transitions, such as fast forward and slow motion, which are usually achieved by frame dropping or frame repetition. The cost of such an operation may be different from that of the repetitive use of the insert or delete operations.

By simply making $t > p$, we obtain the fission (split) operation. In general, however, $t < p$, and in practice we can accomplish both the fusion and fission operations by use of only the fusion operator. That is, before any symbol is used in the edit operation, we look ahead to check if it can be fused with the adjacent symbols. On the other hand, we can prescan the database and query strings and apply the fusion operator where the adjacent symbols meet the criteria for a fusion. The symbols resulting from such a fusion operation are then marked for the purpose of cost and distance computations. Thus, unlike the other edit operations which only need to be applied dynamically on only one of the strings, the fusion operation can be applied to both strings and this can be done offline, before starting the comparison. In the subsequent sections, we shall use only fusion for the fusion/fission operation.

BREAK. **Break inserts or deletes a scene break between two adjacent symbols: insert break**, $ab \rightarrow a\$b$; **delete break**, $a\$b \rightarrow ab$ **($ stands for a scene break)**. This is a special edit operation that allows the insertion or deletion of video scene breaks at any point in the sequence. Because of the significance of scene breaks in video sequences, they may not be very accurately modeled as an ordinary concatenation, or by mere insertion and deletion operations.

3.2. The vString Edit Distance

The bases for the vstring representation are the feature values, which are real (continuous) numbers, representing numerical quantities in the video, such as color, angles, or motion. The vstrings, on the other hand, have discrete values, but depend on the original feature values. We call the continuous feature values the *base/primary representation*, and refer to the vstring as the *symbolic/secondary representation*. For the vstrings, we consider the differences due to both the base representation and the symbolic representation. We refer to the former as the base or primary edit distance, and the latter as the secondary or symbolic edit distance. This combination also provides a natural way for handling the fact that the video string could be multidimensional and that symbols in a vstring could be multivalued. Methods for multidimensional pattern matching [16] can then be used to compute the corresponding multidimensional symbolic edit distance, while traditional multidimensional distance metrics can be used on the multidimensional feature values used for the base representation.

DEFINITION 6. For a given symbol, in the vstring, its **symbol value** is a specially assigned numerical value based on the actual feature values. Let Σ_i be the symbol value of the i-th symbol in the alphabet Σ. Since $i = 1, 2, \ldots, |\Sigma|$, the assignment is simply performed by using $\Sigma_i = i$.

We use the actual feature values to derive the vstrings, but we use the symbol values for computing the base distance. The symbol value thus directly depends on the feature value. We shall use feature values and symbol values interchangeably, unless otherwise stated.

3.2.1. vString Symbolic Edit Distance

Using the new operations, we can derive a general distance measure for video strings as follows: Let O_p be the set of edit operations: $O_p = \{$*insertion, deletion, substitution, swap, fusion, break*$\}$, and let α_p be another set containing the respective cost of each edit operation: $\alpha_p = \{\alpha_{ins}, \alpha_{del}, \alpha_{sub}, \alpha_{swa}, \alpha_{fus}, \alpha_{bre}\}$. Also, let S_E denote a sequence of edit operations which transforms A into B: $S_E = \{{}^{a_1}_{b_1}O_1, {}^{a_2}_{b_2}O_2, \ldots, {}^{a_l}_{b_l}O_l\}$, where ${}^{a_i}_{b_i}O_i$ indicates that $a_i \rightarrow b_i$ by edit operation O, ($O \in O_p$), at edit step

[1] Though scaling as used here is related to the notion of *scaled matching* introduced by Amir *et al.* [5], our definition of scaling is different. Here, scaling is only within the symbols in a string, while in [5], scaling was defined on the entire string, i.e. for $A = a_1 a_2 \cdots a_n$, $A^s = a_1^s a_2^s \cdots a_n^s \neq (AA \cdots A) = A$ concatenated s times, where s is the scaling factor.

i; a_i and b_i are the two symbols[2] involved in ith edit operation. We can then determine the symbolic edit distance between A and B, based on the cost of using this particular edit sequence:

$$\begin{aligned} c(S_E) = & \sum_{O_i=insertion} \alpha_{ins} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right) + \sum_{O_i=deletion} \alpha_{del} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right) \\ & + \sum_{O_i=substitution} \alpha_{sub} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right) + \sum_{O_i=swap} \alpha_{swa} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right) \\ & + \sum_{O_i=fusion} \alpha_{fus} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right) + \sum_{O_i=break} \alpha_{bre} d\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right), \end{aligned} \quad (6)$$

where $d(\begin{smallmatrix} a \\ b \end{smallmatrix} O_l)$ is a distance function whose result depends primarily on the edit operation O_l. The result could also be made to depend on a and b. If the distance is independent of a and b, the symbolic distance defaults to the traditional edit distance, incorporating the new edit operations. Since we will typically have different sequences of edit operations that can transform A into B, we will equally have a set of such edit sequences: $S_{A\to B} = \{S_E^1, S_E^2, \ldots, S_E^q\}$, where S_E^i is the ith edit sequence that transforms A into B. The symbolic edit distance between A and B, $d_s(A, B)$ is then given by the minimum cost edit sequence:

$$d_s(A, B) = \min\{c(S_E) \mid S_E \in S_{A\to B}\}. \quad (7)$$

By choosing different cost functions, different values will be obtained for the edit distance. As with ordinary edit distances, if the distance function $d(\begin{smallmatrix} a \\ b \end{smallmatrix} O_l)$ is a metric, the edit distance $d_s(A, B)$ between strings A and B will also be a metric.

3.2.2. vString Base Edit Distance

The symbolic edit distance is a simple extension from the traditional edit distance, but with consideration of the special vstring edit operations. On its own, it cannot capture important differences in visual content. The base edit distance, on the other hand, uses the feature values and, thus, can capture important content information. The base edit distance can be calculated using the minimum cost edit sequence used to obtain the symbolic edit distance. That is, computing the base edit distance need only be performed *after* computation of the symbolic edit distance. Thus, computation of the base edit distance will not add to the complexity of the symbolic edit distance, since, at worst, only $O(m)$ additional computations will be required.

On the other hand, the minimum-cost edit path for the symbolic distance may not necessarily lead to a minimum cost path for the base distance. Thus, we need to search for the minimum cost path for the base distance, though using a similar procedure. This will, however, lead to an additional computational cost in $O(nm)$. Let $S_{A\to B_{\min}}$ represent the minimum cost edit sequence that transforms A into B using the base edit distance. $S_{A\to B_{\min}}$ is an ordered sequence of edit operations: $S_{A\to B_{\min}} = \{\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix} O_1, \begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix} O_2, \ldots, \begin{smallmatrix} a_l \\ b_l \end{smallmatrix} O_l\}$. We define the base distance between A and B based on the above minimum cost edit sequence:

$$d_b(A, B) = \sum_{i=1}^{l} d_p\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right). \quad (8)$$

More generally, we can express the base distance using the general Minkowsky metric:

$$d_b(A, B) = \left[\sum_{i=1}^{l} \left[d_p\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i\right)\right]^p\right]^{1/p}, \quad (9)$$

where $d_p(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix} O_i)$ is a distance function that uses the feature values to compute the primary distance between symbols. Since the symbol (feature) values are used, the result depends not only on the specific edit operation, but also on the symbols involved. We use d_p to denote the primary distance betwen *symbols* in the vstring, while d_b denotes the base (primary) distance between *vstrings*.

DEFINITION 7. Let $f_v(x)$ be the **symbol value for symbol** x. Let $X : x_1x_2\cdots x_z$ be a string. The *average symbol value* **for string** X is defined as

$$\tilde{f}_v(X) = \frac{1}{z}\sum_{i=1}^{z} f_v(x_i). \quad (10)$$

An analogous definition can also be made using the original feature values. The average is then transformed into a symbol value using the alphabet size. We use Definition 7 to derive the primary distance between symbols $d_p(\cdot)$. For each of the vstring edit operations, d_p is defined as

$$d_p(a, b) = \begin{cases} K_I + |f_v(b) - \tilde{f}_v(B)| & \text{(insertion)} \\ K_D + |f_v(a) - \tilde{f}_v(A)| & \text{(deletion)} \\ |f_v(a) - f_v(b)| & \text{(substitution)} \\ K_W + |f_v(a) - f_v(b)| & \text{(swap)} \\ K_F + |f_v(a) - f_v(b)| & \text{(fusion),} \end{cases} \quad (11)$$

where K_W, K_B, K_D, K_F, and K_I are constants.

If a and b are symbol-blocks, rather than single symbols, i.e. $a = a_1a_2\cdots a_s$, $b = b_1b_2\cdots b_s$, then we define the base distance for the block-swap edit operation as

$$d_p(a, b) = K_w + \sum_{i=1}^{s} |f_v(a_i) - f_v(b_i)| \quad \text{(block-swap).} \quad (12)$$

The above definition for is quite general. The constants can be assigned the value of zero when desired. For instance, we may not need to assign any other special cost after the fusion

[2] They could also be symbol blocks for certain edit operations, for instance in the fusion or the swap edit operation.

(or fission) operation, since we will need to apply other edit operations on the symbols. The base distance for the *insertion* or *deletion* operation depends on the symbols involved. The idea is that, if for instance we insert a symbol into a string X, the distance should be small if the inserted symbol is similar to the average symbol in X. We should have a larger distance if the inserted symbol is very different from the rest of the symbols in X. The substitution operation has no further cost attached to it since it does not require extra operations on the symbols. The break operation is not explicitly indicated in (11). Typically, d_b and d_s will be used together; thus, the symbol distance will also account for the break operation. However, if (11) is to be used independently, the break operation will need to be indicated explicitly. As always, the cost of the special edit operations should be no larger than the equivalent cost of using the traditional edit operations to perform the required transformation. Appendix A shows how to compute the extra cost involved in using the swap operation.

A comment is in order on distances involving the fusion operation. The distance computation uses the fusion part only if either a or b is the result of a fusion operation. For example, if the substitution operation involves a symbol that is the result of a previous fusion operation, an extra cost K_F is incurred. Obviously, if the computation branches into the fusion part; then $(a, b) \neq (\varepsilon, \varepsilon)$. If $a = \varepsilon$, then $f_v(a)$ is replaced with $\tilde{f}_v(B)$, Conversely, if $b = \varepsilon$, $f_v(b)$ is replaced with $\tilde{f}_v(A)$. The constants can be chosen based on the specific application. However, we should have $K_F \leq K_D$, $K_F \leq K_I$ for the fusion operation to be meaninful. (See sections on normalization and experiments.) Also, (11) implies that, apart from the additional fusion cost, K_F, no other attention is paid to the actual input strings involved in the fusion operation. In a later section, we present an alternative method that computes the distance with consideration of the input/output strings.

3.2.3. vString Edit Distance

The final *vstring edit distance* $D_v(A, B)$ is obtained by combining the symbolic edit distance and the base edit distance, through a weighting function,

$$D_v(A, B) = \beta \cdot d_s(A, B) + (1 - \beta) \cdot d_b(A, B), \qquad (13)$$

where β is a weighting function ($0 \leq \beta \leq 1$) which biases the weight to either the symbolic or the base distances. The distances d_s and d_b are suitably normalized. The appropriate choice for the weighting function is still an issue that has to be considered, but it can generally be application dependent or it can be given as an option for the user to decide. The vstring edit distance described above may/or may not be a metric. However, the following lemma holds.

LEMMA 2. *If d_s and d_b are metric distances, then the vstring edit distance $D_v(A, B)$ between two video strings A and B is also a metric. That is, $D_v(A, A) = 0$, $D_v(A, B) \geq 0$, $D_v(A, B) = D_v(B, A)$, $D_v(A, B) \leq D_v(A, C) + D_v(C, B)$.*

4. ALTERNATIVE DISTANCE COMPUTATION FOR MATCHING WITH THE FUSION OPERATION

We can apply the fusion operation offline on both the query and the database strings before performing the matching, while keeping information on the parameters of the operation (p and t). The vstring distance can then be computed by manipulating the parameters and by use of the internal representation of the modified vstrings. We assume that t is fixed at $t = 1$, while p can vary. Thus, internally, we use the length difference between the input and output strings as part of the internal representation. That is, for the simple transformation performed by the fusion operation, $a^x \rightarrow a^y$, we represent the result as $a^{|x-y|}$.

EXAMPLE 4. Table 2 shows the internal representation for the results of the fusion operation on sample input strings.

To compute the cost of the fusion operation, we use the above internal representation by making the following considerations:

1. The fusion operation can be realized by repetitive insertion or deletion.
2. Matching with fusion should result in smaller distances, as compared with the distances obtained without fusion (i.e., by use of, say, repeated deletion).
3. The larger the difference between the length of the input and output strings (x and y), the larger the cost. The difference also depends on the p and t parameters.

The first two considerations imply that the cost of the fusion operation can be made a function of the cost for insertion and/or deletion. The last point was not considered by Eq. (11). Below we give an alternative method for computing the distance, based on the internal representation.

Let: $A : a_1 a_2 \cdots a_n$ and $B : b_1 b_2 \cdots b_m$ be the initial vstrings. Let $A_f : c_1^{n_1} c_2^{n_2} \cdots c_{n_f}^{n_{n_f}}$ and $B_f : d_1^{m_1} d_2^{m_2} \cdots d_{m_f}^{m_{m_f}}$ be their corresponding international representations, where c_i is the ith symbol in A_f, n_i is the difference between the lengths of input and output strings from which c_i was formed, and $n_f = |A_f|$ is the length of the new string; and similarly for B_f. Also, let K_f be the new cost for the fusion operation using the alternative computation, $K_f \leq K_D$, $K_f \leq K_I$. The K_f used here is not necessarily equal to the K_F in (11). In the last section, the distance

TABLE 2
Results of the Fusion Operation and Their Internal Representation ($p = 3, t = 1$)

	String1	String2	String3
Original string	*aabbbc*	*aabbbbc*	*aaaabbbbbccc*
String after fusion	*abc*	*abbc*	*aabbc*
Internation representaion	$a^1b^2c^0$	$a^1b^2b^0c^0$	$a^2a^0b^2b^1c^2$

between A_f and B_f was computed using only the values for c_i and d_j, while ignoring the length difference between the input and output strings. A better measure of the distance should include information on the parameters of the fusion operation. For the discussion here, we assume $K_f = K_D$ and $p = 3, t = 1$.

We observe that $\tilde{f}_v(A)$ and $\tilde{f}_v(B)$ can be computed directy from the internal representation A_f and B_f, respectively. We use the following equivalent definition for the average symbol value for the original string A, using the internal represention A_f:

$$\tilde{f}_v(A) = \tilde{f}_v(A_f) = \frac{\sum_{i=1}^{n_f}(n_i+1)c_i}{\sum_{i=1}^{n_f}(n_i+1)}. \tag{14}$$

To obtain the distance between two symbols $c_i^{n_i}$ from A_f and $d_j^{m_j}$ from B_f, we need to consider five cases:

Case I. Simple case of repeatitive substitution:

$$\begin{aligned} &n_i = m_j, \quad c_i \neq \varepsilon, \quad d_j \neq \varepsilon, \\ &d\left(c_i^{n_i},\ d_j^{m_j}\right) = (n_i+1)|\,f_v(c_i) - f_v(d_j)|. \end{aligned} \tag{15}$$

Case II. Trivial case of insertion:

$$\begin{aligned} &n_i \neq m_j, \quad c_i = \varepsilon, \quad d_j \neq \varepsilon, \\ &d\left(c_i^{n_i}, d_j^{m_j}\right) = (m_1+1)(K_f + |\,f_v(d_j) - \tilde{f}_v(B_f)|). \end{aligned} \tag{16}$$

Case III: Trivial case of deletion:

$$\begin{aligned} &n_i \neq m_j, \quad c_i \neq \varepsilon, \quad d_j = \varepsilon, \\ &d\left(c_i^{n_i}, d_j^{m_j}\right) = (n_i+1)(K_f + |\,f_v(c_i) - \tilde{f}_v(A_f)|). \end{aligned} \tag{17}$$

Case IV. More difficult case of insertion or deletion:

$$\begin{aligned} &n_i \neq m_j, \quad c_i = d_j, \\ &d\left(c_i^{n_i}, d_j^{m_j}\right) \\ &= \begin{cases} |n_i - m_j| \cdot (K_f + |\,f_v(c_i) - \tilde{f}_v(A_f)|), & \text{if } n_i > m_j, \\ |n_i - m_j| \cdot (K_f + |\,f_v(d_i) - \tilde{f}_v(B_f)|), & \text{if } n_i < m_j. \end{cases} \end{aligned} \tag{18}$$

Case V. Most difficult case, requring substitution and insertion and/or deletion:

$$\begin{aligned} &n_i \neq m_j, c_i \neq d_j, c_i \neq \varepsilon, d_j \neq \varepsilon, \\ &d\left(c_i^{n_i}, d_j^{m_j}\right) = (1 + \min\{n_i, m_j\}) \cdot (|\,f_v(c_i) - f_v(d_j)|) \\ &+ \begin{cases} |n_i - m_j| \cdot (K_f + |\,f_v(c_i) - \tilde{f}_v(A_f)|), & \text{if } n_i > m_j, \\ |n_i - m_j| \cdot (K_f + |\,f_v(d_i) - \tilde{f}_v(B_f)|), & \text{if } n_i < m_j. \end{cases} \end{aligned} \tag{19}$$

TABLE 3
Example Parameter Values for Different Cases of the Fusion Operation ($p = 3, t = 1$)

Case	Description	Original strings	After fusion	c_i, d_j	n_i, m_j
I	$n_i = m_j; c_i \neq \varepsilon; d_j \neq \varepsilon$	$A : aaa$	$A_f : a^2$	$c_i = a$	$n_i = 2$
		$B : bbb$	$B_f : b^2$	$d_j = b$	$m_j = 2$
II	$n_i \neq m_j; c_i = \varepsilon; d_j \neq \varepsilon$	$A : \varepsilon$	$A_f : \varepsilon$	$c_i = \varepsilon$	$n_i = -1$
		$B : bbb$	$B_f : b^2$	$d_j = b$	$m_j = 2$
III	$n_i \neq m_j; c_i \neq \varepsilon; d_j = \varepsilon$	$A : aaa$	$A_f : a^2$	$c_i = a$	$n_i = 2$
		$B : \varepsilon$	$B_f : \varepsilon$	$d_j = \varepsilon$	$m_j = -1$
IV	$n_i \neq m_j; c_i = d_j$	$A : aaa$	$A_f : a^2$	$c_i = a$	$n_i = 2$
		$B : aa$	$B_f : a^1$	$d_j = a$	$m_j = 1$
V	$n_i \neq m_j; c_i \neq d_j;$	$A : aaa$	$A_f : a^2$	$c_i = a$	$n_i = 2$
	$c_i \neq \varepsilon; d_j \neq \varepsilon;$	$B : bb$	$B_f : b^1$	$d_j = b$	$m_j = 1$

The above cases are sumarized in Table 3 with example strings. In the table, values of -1 were used in the internal representation of the null symbol. This is simply to distinguish it from other symbols which could have zero in their internal representation. From (15) and (16), this will have no effect on the distance.

Notice that the above distance computation always selects the minimum cost path. With the above method, the fusion cost will no longer appear as an explict edit cost, but will be embedded in the other operations. More importantly, we can also use the above relations for the symbolic distance. We recall that the symbolic distance treats the symbols as binary (presence/absence) symbols. Thus, to use the alternative computation for the symbol distance we make use of the definitions:

$$f_v(c_i) - f_v(d_j) = \begin{cases} 0, & \text{if } c_i = d_i, \\ 1, & \text{if } c_i \neq d_i; \end{cases} \tag{20a}$$

$$f_v(x) - \tilde{f}_v(X) = 0. \tag{20b}$$

Since the time required for string matching is in O(nm), the main advantage in using the alternative computation is the improved speedup in the matching process. Typically, $n_f \leq n$ and $m_f \leq m$, implying that shorter strings are manipulated by the alternative method. The above internal representation is used to keep the presentation simple. The representation is neither optimal in space nor in time. Thus, there is room for some improvement, for instance by increasing the value of p. A better approach could be to enforce a constraint, $c_i \neq c_{i+1}$, in which no two adjacent symbols in the internal representation will be the same. Here, a given symbol c_i in A_f can be internally represented as $c_i^{x_i, y_i}$, where x_i and y_i now explicitly express the length of the input and output symbols. An alternative computation using the new representation can be derived and used to compute the edit distances. It is interesting to observe the significance of the alternative distance computation. The internal representation is essentially a form of encoding on the video strings. This means that the method can equally be applied on encoded sequences, such as run-length encoded sequences.

5. NORMALIZATIONS

For the normalizations below, we assume equal costs for the three traditional edit operations. With unequal costs, different normalizations may be needed. For the vstring symbol distance d_s, the major normalization required is that against differences in sequence lengths. We use two normalizations on the symbol edit distance:

$$\textit{NORM1}: \quad d_{s1} = \frac{d_s - ||A| - |B||}{\min\{|A|, |B|\}} \tag{21}$$

$$\textit{NORM2}: \quad d_{s2} = \frac{d_s}{\max\{|A|, |B|\}}. \tag{22}$$

NORM1 emphasizes the *existence* of similar subsequences between the database and the query, while minimizing the effect of length differences. The smaller the distance between the sequences, the longer the similar subsequence(s)—hence, the more the overall similarity. Matching with *NORM1* will thus result in better recall, but less precision. Hypothetically, retrieval based only on *NORM1* could return a one-frame sequence for a 1000-frame query, where the 1000 frames are all copies of the same frame.

NORM2 accounts for both *content-similarity* and *length differences*. The results are more precise (in terms of both visual content and sequence duration), but *NORM2* could miss out possibly similar sequences (say, in terms of visual content) purely on account of length differences.

With the vstring base distance, we perform a transformation into symbols with the actual feature values, but use the symbol values for matching. Therefore, the symbol value depends on the alphabet size. The vstrings thus needs to be normalized for different alphabet sizes. For the ith symbol in an alphabet Σ, the symbol value is given by $\Sigma_i = 1, 2, \ldots, |\Sigma|$. We normalize the symbol values as $f_v(\Sigma_i) = \Sigma_i/(|\Sigma| - 1)$. Thus, the symbol values fall in the range of $1/(|\Sigma| - 1)$ to $1 + 1/(|\Sigma| - 1)$ and not necessarily from 0 to 1.

Like the symbol distance, the base distance is also normalized against differences in sequence lengths. The major issue here is that, unlike the symbol distance, the primary distance between two symbols will not necessarily be 1 but will depend on their symbol values. Thus, we need to adjust the sequence lengths (and their differences) accordingly. For the base distance d_b, the corresponding normalizations are:

$$\textit{NORM1}: \quad d_{b1} = \frac{d_b - ||A| - |B|| \cdot (|\tilde{f}_v(A) - \tilde{f}_v(B)| + K_I)}{\min\{|A|, |B|\} \cdot \max\{\tilde{f}_v(A), \tilde{f}_v(B)\}}, \tag{23}$$

$$K_I = \frac{1}{|\Sigma| - 1},$$

$$\textit{NORM2}: \quad d_{b2} = \frac{d_b}{\min\{|A|, |B|\} \cdot \max\{\tilde{f}_v(A), \tilde{f}_v(B)\}}. \tag{24}$$

K_I is used to remove the additional cost incurred by the extra deletions or insertions that will be needed on sequences that are of differing lengths. $K_I = K_D$ are the same as the constants used in see Section 3. The constants decrease with increasing alphabet size. This is logical since the distance will typically increase with increasing alphabet size. Thus, if used as defined above, the constants will also serve to reduce the effect of the alphabet size.

In subsequent sections, we assume d_s and d_b are in the normalized form, unless otherwise stated.

6. EXPERIMENTS

To check the effectiveness of the proposed distance measure, we carried out some tests using real video sequences. The experiments were carried out in a MATLAB environment, running on UltraSparc workstations. Two color features from the frames (the average color and the standard deviation) were used as the indexing features. Each frame in the sequence was subdivided into η subframes, and for each subframe two different vstrings were formed using the two features. The vstring edit distances (d_s and d_b,) were then computed independently for each feature and for each subframe, after which the results were combined. For the experiments, we used the unit cost for each edit operation: $\eta = 9$, except for the case of studying the effect of η; and $|\Sigma| = 4$, except when studying the effect of $|\Sigma|$. For all experiments, we used the actual feature values to derive the vstrings and then used the symbol values to calculate the base distances.

First, we studied the behavior of the vstring base and symbol distances under the two methods of normalization (*NORM1* and *NORM2*) for different settings of the alphabet size $|\Sigma|$, and number of subframes, η. Information about such behavior is required for parameter setting and for fine-tuning the results. For instance, we need to determine appropriate scaling factors and weights for combining say d_s and d_b (or *NORM1* and *NORM2*) results. We took different sequences and selected 12 scenes from each sequence. For each sequence, the vstring distance between the 12 scenes was computed. The scenes were selected to include both similar and nonsimilar scenes in the sequence.

Figure 1 shows the variation of the distances with the alphabet size for different distance normalizations. For both *NORM1* and *NORM2* the symbol distance d_s (Figs. 1a and b) was found to be very sensitive to $|\Sigma|$, increasing quite rapidly with the alphabet size. This is expected since the probability of having a match decreases with increasing alphabet size. On the other hand, the base distance (Figs. 1c and d) is more stable; though it also changes with $|\Sigma|$, the change is much less when compared with the case of d_s. This is a more preferable behavior since the underlying similarity (or distance) between two scenes should not be greatly affected by small changes in the alphabet size. The figure also shows that, under *NORM2*, d_b decreases with increasing alphabet size. Though less obvious, this is due to the effect of K_I and K_D ($K_I = K_D = 1/(|\Sigma| - 1)$). For two given sequences, the length difference is constant, but the additional cost due to the extra

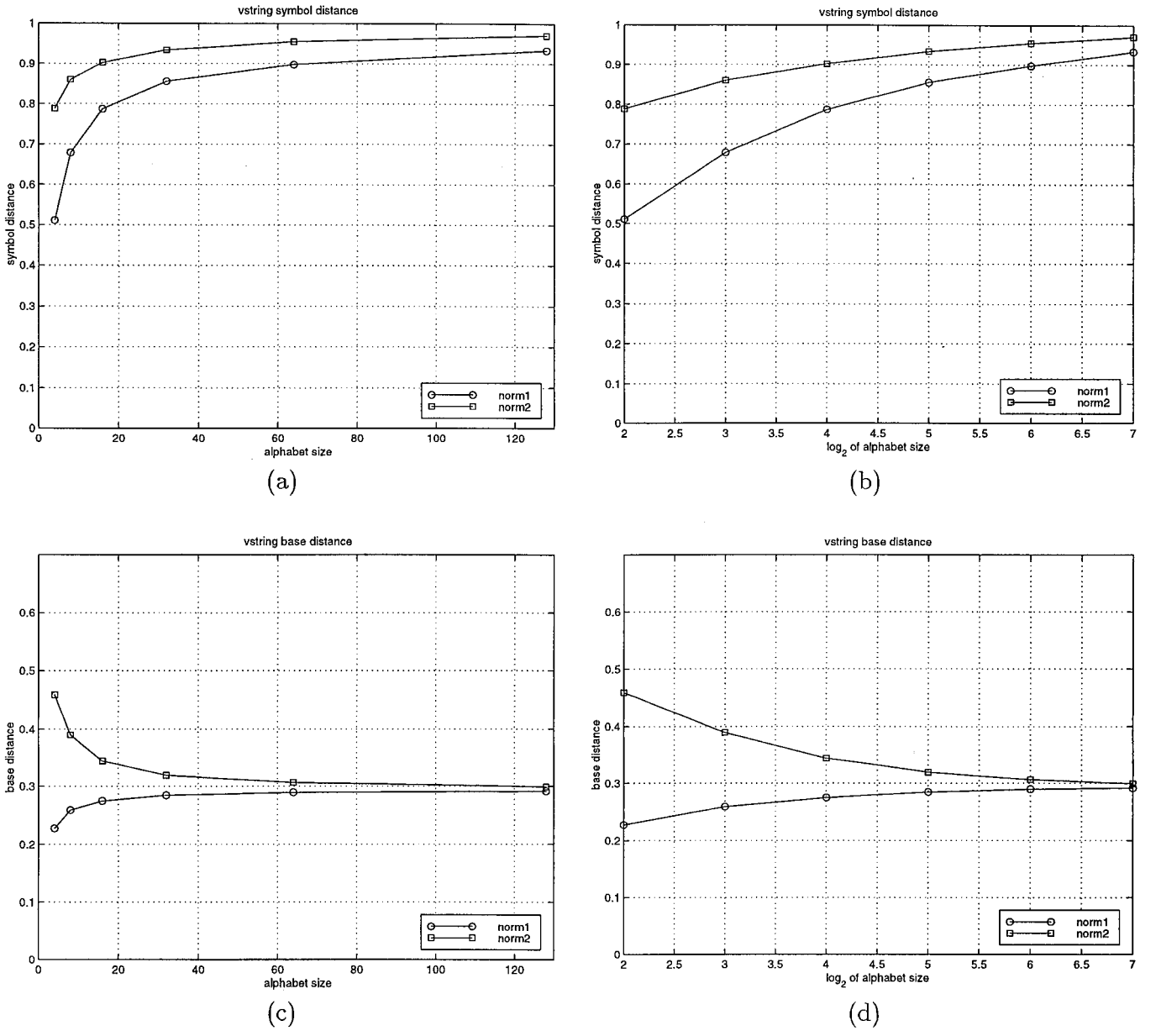


FIG. 1. Variation of vstring edit distances with alphabet size, for two distance normalizations: (a, b) symbolic edit distances; (c, d) base edit distance.

insertion/deletion needed to transform one sequence into the other is inversely proportional to $|\Sigma|$. As expected, for a given alphabet size the distances obtained using *NORM2* are always greater than those from *NORM1*. Similarly, the symbol distances are generally greater than the base distances. The same variations are shown on a log scale, indicating that for a given η the vstring distances tend to change more linearly with the log of $|\Sigma|$, rather than with $|\Sigma|$.

Figure 2 shows the variation of the distances for varying number of subframes, η, for different values of $|\Sigma|$. We expected the distance to increase with increasing η, since increasing the number of subframes should imply that more specific information is being used to match the sequences. However, it was observed that for a given alphabet size the distances were generally stable for different values of η. Though d_b increased slightly with increasing η, the increase is negligible (typically less than 0.05 for the values of $|\Sigma|$ tested). The implication is that, with respect to the color features used—(the average color and the standard deviation), η is not a critical issue. We do not think that this will be the case in general; it could be different for other features, such as motion. Judging from the results shown in the figure, we can conveniently use any value for η (we suggest $\eta \geq 9$) without affecting the results. This is an important consideration (especially for speed), since more time will be required to compute the combined distance (from the η vstrings for each feature used) as η gets large. On the other hand, for a given η, d_s increases

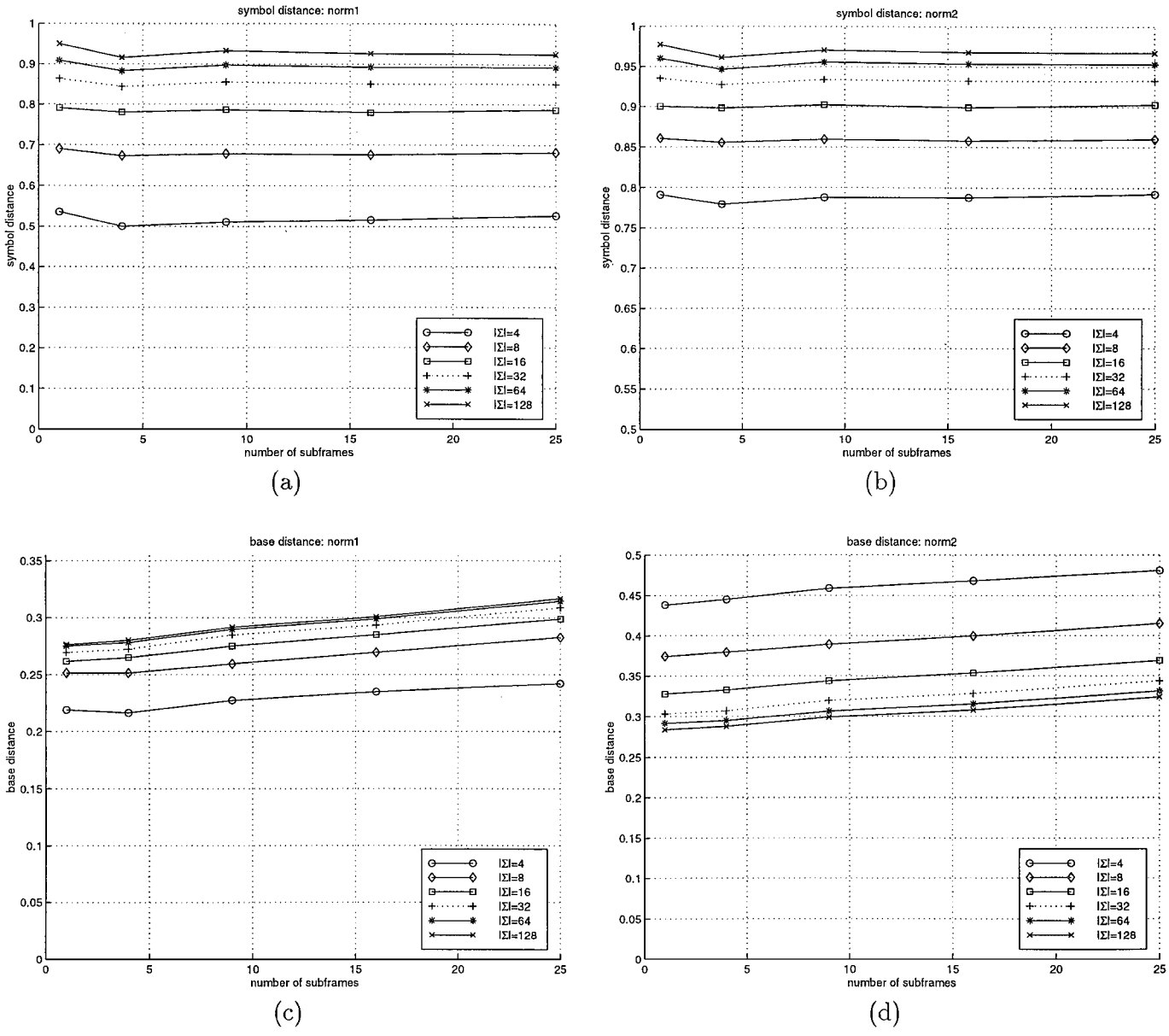


FIG. 2. Variation of vstring base and symbol edit distances with number of subframes for different alphabet sizes using two distance normalizations.

for both *NORM1* and *NORM2* while d_b increases for *NORM1*, but decreases for *NORM2* as $|\Sigma|$ increases. This mirrors the observations from Fig. 1, but for different values of η.

To evaluate the vstring distance measure, we then compared the results returned by the proposed methods by visually comparing different video sequences for similarity. First, we used the same sequences used in studying the behavior of the distances (i.e. the 12 specially selected scenes for each sequence). For each sequence each of the 12 scenes was matched against the other 11 scenes in the sequence. Figure 3 shows the representative frames (selected periodically from each scene) for one of the sequences. The sequence is taken from a news video (a relatively simple video), and thus, it is easy to visually compare the scenes.

The distance results are shown in Tables 4 and 5 for *NORM1*. Table 4 shows the vstring distance matrix for the 12 scenes, while Table 5 gives the ranking of the scenes based on their similarity—using the vstring distance. The indicated distances are the combined distances obtained from d_s and d_b, using $\beta = 0.5$. For $|\Sigma| = 4$ we multiplied d_b with an initial scaling factor of 1.75 before combing with d_s. This was informed by the average values of d_s and d_b for a given $|\Sigma|$ for the different normalizations. It can be seen that the length differences have no effect on the results. Equivalent results for *NORM2* are shown in Tables 6 and 7. No further scaling was used on d_s and d_b, since their values are comparable at $|\Sigma| = 4$. The *NORM2* results are, however, very much influenced by the differences in scene lengths. (Compare the ranking of v1 versus v11 and v12 in Table 3 with those in

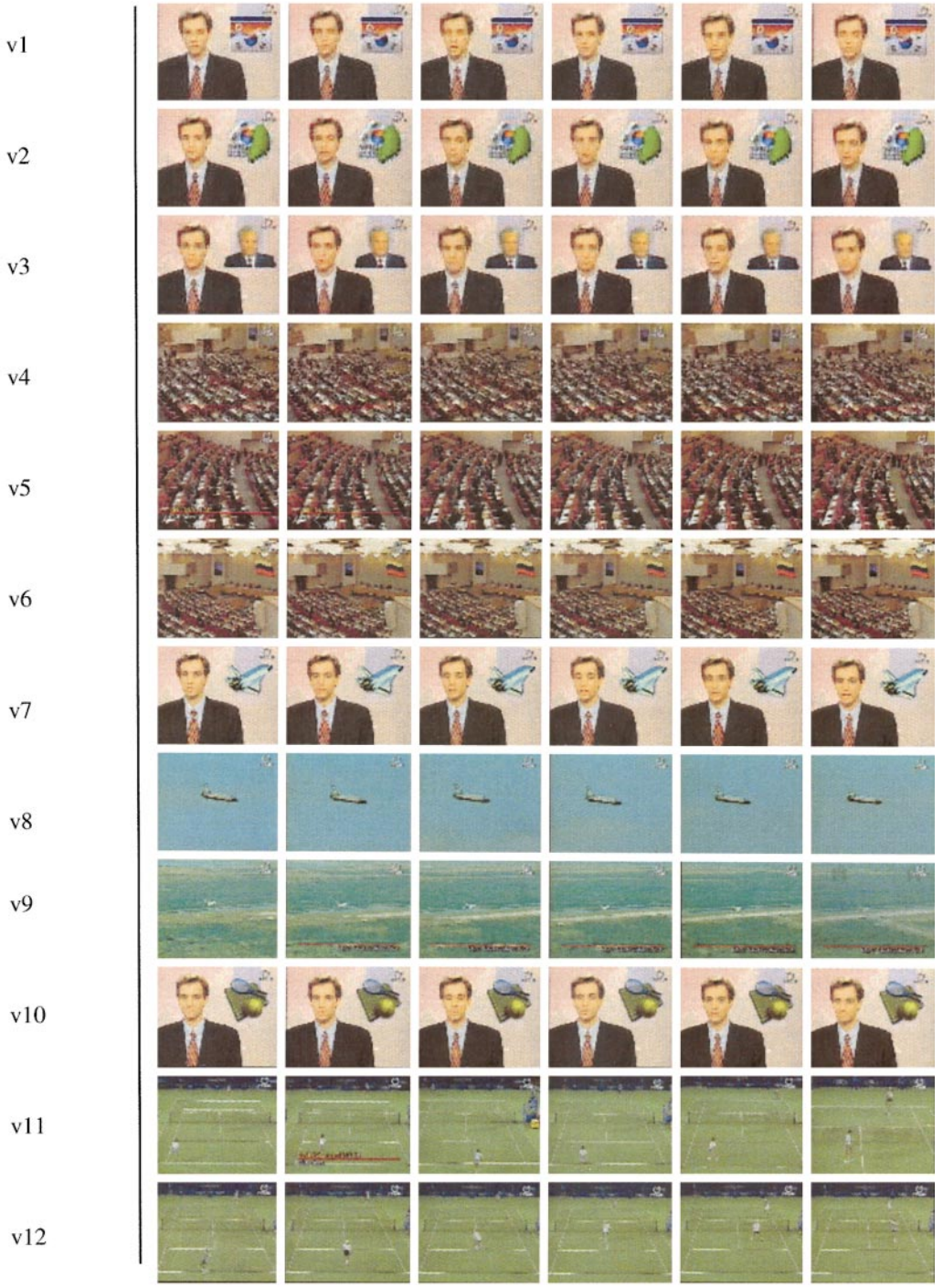


FIG. 3. Representative frames for 12 *specially* selected scenes for one of the video sequences used in testing. Scenes are selected such that, both similar and nonsimilar sequences will be included. The scene lengths are given in the tables. The indicated scenes are taken from a news video—a relatively simple sequence and, thus, easy to compare.

FIG. 4. Three representative frames for 24 *randomly* selected scenes from six different video sequences. From each sequence, four scenes were selected randomly, without regard to their similarity. The scenes are arranged (and can be grouped) according to their original source: ADVERT-1, v1–v4; NEWS, v5–v8; COMMERCIAL-FILM-1, v9–v12; COMMERCIAL-FIRM-2, v13–v16; ADVERT-2, v17–v20; DOCUMENTARY, v21–v24. The scene lengths are given in the tables (see text).


v1
v2
v3
v4
v5
v6
v7
v8
v9
v10
v11
v12
v13
v14
v15
v16
v17
v18
v19
v20
v21
v22
v23
v24

TABLE 4
vString Distance Matrix for the Video Sequences (Using *NORM*1) Shown in Fig. 3

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Len[a]	155	386	563	53	85	38	121	46	110	414	434	133
v1	0	0	0	0	0	0	0	0	0	0	0	0
v2	0.09	0	0	0	0	0	0	0	0	0	0	0
v3	0.11	0.06	0	0	0	0	0	0	0	0	0	0
v4	0.59	0.61	0.59	0	0	0	0	0	0	0	0	0
v5	0.66	0.66	0.64	0.12	0	0	0	0	0	0	0	0
v6	0.56	0.60	0.57	0.12	0.16	0	0	0	0	0	0	0
v7	0.04	0.13	0.10	0.66	0.69	0.63	0	0	0	0	0	0
v8	0.61	0.69	0.68	0.57	0.54	0.56	0.63	0	0	0	0	0
v9	0.63	0.62	0.59	0.33	0.36	0.31	0.67	0.35	0	0	0	0
v10	0.09	0.04	0.05	0.54	0.61	0.51	0.09	0.62	0.57	0	0	0
v11	0.66	0.65	0.64	0.29	0.30	0.28	0.66	0.46	0.32	0.64	0	0
v12	0.73	0.70	0.68	0.31	0.34	0.33	0.75	0.51	0.38	0.67	0.03	0

[a] Len stands for length of the video sequence.

TABLE 5
Ranking of the Video Sequences in Descending Order of Similarity—Based on the Vstring Distance

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Len	155	386	563	53	85	38	121	46	110	414	434	133
Rank												
1	1	2	3	4	5	6	7	8	9	10	11	12
2	7	10	10	5	4	4	1	9	6	2	12	11
3	10	3	2	6	6	5	10	11	11	3	6	4
4	2	1	7	11	11	11	3	12	4	1	4	6
5	3	7	1	12	12	9	2	5	8	7	5	5
6	6	6	6	9	9	12	8	6	5	6	9	9
7	4	4	9	10	8	10	6	4	12	4	8	8
8	8	9	4	8	10	1	4	1	10	9	10	10
9	9	11	5	1	3	8	11	10	3	5	3	3
10	5	5	11	3	1	3	9	7	2	8	2	2
11	11	8	8	2	2	2	5	3	1	11	1	1
12	12	12	12	7	7	7	12	2	7	12	7	7

Table 5. Visually, v11 and v12 look similar, but differ in scene duration). Typically for both d_s and d_b, *NORM2* distances are always greater than the corresponding *NORM1* distances.[3] While *NORM1* provides results that are visually more intuitive, the advantage of using *NORM2* is that any two sequences that it proposes as similar (the first few positions in the ranking) will be similar in both *duration* and *visual content*. *NORM2* thus provides a more accurate result, but at the risk of missing out some scenes that are visually similar, but which differ only in scene duration.[4]

The technique was then used on a more difficult set of video sequences. Here, four scenes were randomly selected from six different video sequences (for a total of 24 scenes). (The scenes were not involved in the previous test with 12 scenes). Each of the 24 scenes were then matched against the remaining 23 scenes in the set. This is a controlled simulation of the practical situation where one will wish to search for a given video scene in a video database, which typically contains video sequences from different sources. Figure 4 shows the representative frames for each scene. The scenes are arranged (and can be grouped) according to their original source: ADVERT-1, V1–V4; NEWS, V5–V8; COMMERCIAL-FILM-1, V9–V12; COMMERCIAL-FIRM-2, V13–V16; ADVERT-2, V17–V20; DOCUMENTARY, V21–V24. Scenes belonging to the same group are not necessarily similar.

The results are shown in Tables 8 and 9. We take visual similarity to be more important than length differences, and thus

TABLE 6
vString Distance Matrix (Using *NORM*2) for the Video Sequences Shown in Fig. 3

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Len	155	386	563	53	85	38	121	46	110	414	434	133
v1	0	0	0	0	0	0	0	0	0	0	0	0
v2	0.45	0	0	0	0	0	0	0	0	0	0	0
v3	0.54	0.25	0	0	0	0	0	0	0	0	0	0
v4	0.71	0.78	0.79	0	0	0	0	0	0	0	0	0
v5	0.67	0.77	0.79	0.33	0	0	0	0	0	0	0	0
v6	0.73	0.79	0.79	0.29	0.48	0	0	0	0	0	0	0
v7	0.17	0.52	0.57	0.70	0.66	0.73	0	0	0	0	0	0
v8	0.73	0.82	0.83	0.51	0.62	0.53	0.71	0	0	0	0	0
v9	0.61	0.74	0.77	0.55	0.41	0.61	0.58	0.59	0	0	0	0
v10	0.47	0.07	0.22	0.77	0.76	0.78	0.52	0.81	0.74	0	0	0
v11	0.74	0.56	0.60	0.75	0.71	0.77	0.76	0.81	0.69	0.53	0	0
v12	0.63	0.76	0.79	0.59	0.48	0.66	0.63	0.71	0.40	0.76	0.55	0

TABLE 7
Ranking of the Video Sequences in Descending Order of Similarity Using *NORM*2 Distance

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
Len	155	386	563	53	85	38	121	46	110	414	434	133
Rank												
1	1	2	3	4	5	6	7	8	9	10	11	12
2	7	10	10	6	4	4	1	4	12	2	10	9
3	2	3	2	5	9	5	10	6	5	3	12	5
4	10	1	1	8	12	8	2	9	4	1	2	11
5	3	7	7	9	6	9	3	5	7	7	3	4
6	9	11	11	12	8	12	9	12	8	11	9	7
7	12	9	9	7	7	1	12	7	1	9	5	1
8	5	12	12	1	1	7	5	1	6	12	1	6
9	4	5	5	11	11	11	4	11	11	5	4	8
10	6	4	4	10	10	10	8	10	10	4	7	10
11	8	6	6	2	2	2	6	2	2	6	6	2
12	11	8	8	3	3	3	11	3	3	8	8	3

[3] Some values in Table 4 (*NORM2*) are smaller than the corresponding values in Table 2 (*NORM1*) due to the scaling factor used on the results from *NORM1*.

[4] A comment is in order on the use of *NORM1* and *NORM2* results. We can simply take a weighted average of the results from the two normalization methods. This will, however, lose information about the cause of any differences—if the sequences are not similar. Since *NORM1* could lead to no misses (with respect to visual content as compared to results from *NORM2*), a better method for combining the two results could be to use *NORM1* results for the initial selection of probable (visually)similar scenes and then to use *NORM2* results to verify if the scenes are also similar in duration.

TABLE 8

vString Distance Matrix (Using *NORM*1) for the Video Sequences Shown in Fig. 4

	v1	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v8**	**v9**	**v10**	**v11**	**v12**	**v13**	**v14**	**v15**	**v16**	**v17**	**v18**	**v19**	**v20**	**v21**	**v22**	**v23**	**v24**
Len	24	34	22	36	125	187	77	87	79	115	55	48	77	274	41	73	34	111	33	53	372	50	111	55
v1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v2	0.31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v3	0.24	0.35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v4	0.31	0.45	0.18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v5	0.45	0.31	0.37	0.42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v6	0.46	0.32	0.43	0.43	0.26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v7	0.39	0.21	0.41	0.42	0.33	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v8	0.47	0.45	0.52	0.53	0.68	0.55	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v9	0.42	0.36	0.40	0.47	0.54	0.51	0.47	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v10	0.31	0.24	0.32	0.32	0.43	0.38	0.22	0.38	0.42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v11	0.34	0.20	0.31	0.35	0.27	0.30	0.23	0.50	0.38	0.22	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v12	0.47	0.42	0.49	0.46	0.51	0.45	0.36	0.24	0.33	0.22	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0
v13	0.69	0.64	0.61	0.61	0.37	0.46	0.69	0.91	0.78	0.63	0.60	0.78	0	0	0	0	0	0	0	0	0	0	0	0
v14	0.51	0.24	0.35	0.35	0.23	0.35	0.27	0.57	0.51	0.37	0.26	0.43	0.34	0	0	0	0	0	0	0	0	0	0	0
v15	0.46	0.24	0.49	0.48	0.28	0.28	0.23	0.44	0.41	0.22	0.23	0.43	0.64	0.24	0	0	0	0	0	0	0	0	0	0
v16	0.39	0.31	0.37	0.40	0.36	0.32	0.35	0.58	0.53	0.33	0.26	0.54	0.61	0.19	0.30	0	0	0	0	0	0	0	0	0
v17	0.43	0.37	0.33	0.43	0.34	0.43	0.41	0.63	0.46	0.34	0.33	0.57	0.48	0.37	0.42	0.37	0	0	0	0	0	0	0	0
v18	0.36	0.31	0.30	0.27	0.42	0.40	0.35	0.58	0.52	0.42	0.32	0.51	0.57	0.29	0.28	0.32	0.24	0	0	0	0	0	0	0
v19	0.45	0.38	0.37	0.42	0.39	0.46	0.42	0.63	0.47	0.37	0.36	0.59	0.53	0.37	0.48	0.39	0.17	0.24	0	0	0	0	0	0
v20	0.39	0.31	0.30	0.32	0.40	0.40	0.35	0.58	0.51	0.33	0.35	0.52	0.55	0.27	0.32	0.35	0.18	0.16	0.21	0	0	0	0	0
v21	0.82	0.94	0.77	0.75	0.67	0.74	0.91	0.94	0.92	0.87	0.82	0.91	0.32	0.66	0.84	0.71	0.72	0.78	0.72	0.77	0	0	0	0
v22	0.47	0.40	0.44	0.50	0.46	0.51	0.51	0.65	0.54	0.42	0.37	0.63	0.60	0.44	0.39	0.32	0.45	0.50	0.47	0.55	0.59	0	0	0
v23	0.51	0.39	0.41	0.38	0.36	0.43	0.40	0.67	0.63	0.45	0.37	0.56	0.50	0.26	0.33	0.35	0.30	0.32	0.33	0.29	0.64	0.46	0	0
v24	0.71	0.68	0.60	0.63	0.65	0.68	0.74	0.81	0.76	0.65	0.62	0.77	0.53	0.47	0.64	0.57	0.46	0.57	0.46	0.56	0.33	0.37	0.51	0

TABLE 9
Ranking of the Video Sequences in Descending Order of Similarity

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v15	v16	v17	v18	v19	v20	v21	v22	v23	v24
Len	24	34	22	36	125	187	77	87	79	115	55	48	77	274	41	73	34	111	33	53	372	50	111	55
Rank																								
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
2	3	11	4	3	14	5	2	12	12	7	2	10	21	16	10	14	19	20	17	18	13	16	14	21
3	4	7	1	18	6	15	10	9	2	11	10	8	14	5	11	11	20	17	20	17	24	11	20	22
4	10	10	20	1	11	7	11	10	8	15	7	9	5	15	7	15	18	19	18	19	22	24	17	19
5	2	15	18	10	15	11	15	7	11	12	15	7	6	2	14	2	23	4	23	14	23	15	18	17
6	11	14	11	20	2	2	14	15	3	2	14	2	17	11	2	18	11	15	11	23	14	2	15	14
7	18	5	10	14	7	16	6	2	15	1	16	15	23	23	6	22	3	14	3	3	5	10	19	23
8	7	16	17	11	17	14	5	1	1	4	5	14	24	20	5	6	5	3	14	2	16	3	16	13
9	20	18	2	23	16	10	18	11	10	3	6	11	19	7	18	10	10	2	10	4	17	14	5	20
10	16	20	14	16	23	20	16	3	17	20	3	6	20	18	16	7	2	11	2	15	19	17	11	16
11	9	1	5	7	3	18	20	4	7	16	18	4	18	13	20	20	14	16	5	10	6	5	4	18
12	17	6	19	5	13	4	12	6	4	17	17	1	22	6	23	23	16	23	16	11	4	23	2	3
13	19	3	16	19	19	3	1	14	19	19	1	3	11	4	22	5	7	7	4	7	3	19	7	11
14	5	9	9	17	20	17	23	20	14	14	20	18	16	3	9	17	15	1	7	16	20	1	3	4
15	6	17	23	6	4	23	17	18	20	6	4	5	4	17	17	3	4	6	1	1	18	18	6	15
16	15	19	7	2	18	12	3	16	6	8	19	20	3	19	12	1	6	10	24	6	1	4	10	10
17	12	23	6	12	10	1	4	19	18	22	22	16	10	10	8	19	1	5	6	5	11	7	22	5
18	22	22	22	9	1	19	19	17	16	18	23	23	15	12	1	4	22	22	22	9	15	6	13	2
19	8	12	15	15	22	13	8	22	5	9	9	17	2	22	4	9	9	12	9	12	10	9	1	6
20	14	8	12	22	12	22	9	23	22	5	12	19	7	24	19	12	24	9	15	13	7	20	24	1
21	23	4	8	8	9	9	22	5	23	23	8	22	1	9	3	24	13	13	13	22	12	21	12	7
22	13	13	24	13	24	8	13	24	24	13	13	24	9	1	13	8	12	24	12	24	9	13	9	9
23	24	24	13	24	21	24	24	13	13	24	24	13	12	8	24	13	8	8	8	8	2	12	21	12
24	21	21	21	21	8	21	21	21	21	21	21	21	8	21	21	21	21	21	21	21	8	8	8	8

only *NORM1* results are indicated. We acknowledge the fact that the distance values returned by the proposed distance measure may not always correspond to what a human observer may assign to the scenes; our experiments were based only on color features. In retrieval involving visual data, however, the ranking of the results are often taken to be more important. We can see from the tables that the rankings returned by the distance measure are quite reliable—the visually similar scenes are always ranked in the first few positions. The tables thus show that the vstring edit distance provides a good measure of the similarity between video sequences.

7. CONCLUDING REMARKS

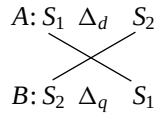
An edit distance based method has been presented for measuring the similarity between video sequences. The vstring edit distance incorporates two basic components that account for the visual similarity between video scenes and for the temporal-constraints in the video. Experimental results have shown that the vstring edit distance is able to rank different video scenes consistently according to their similarity. The vstring edit distance thus provides a reliable measure of the distance between video sequences. It can form the basis for applications where retrieval requires a consideration of the spatio-temporal constraints in the video, beyond the current method of simple image-based retrieval using key frames.

With the proposed measure, the distance between any two vstrings depends very much on the cost of the edit operations and on the number of edit operations. When the cost of each edit operation is assumed to be the same, for instance unity, the problem of choosing a threshold for similarity is reduced to just deciding the k-differences that are to be allowed in the match. As one may have noted from the previous discussions, a uniform cost for each edit operation may not accurately model the importance of each operation in the video sequence. For instance, a deletion operation (analogous to removing one frame in a video scene) should not carry the same weight as a break operation (for instance inserting a scene break) which divides one scene into two different scenes. Parametric edit distances try to put these issues into consideration in determining the suitable cost functions for each edit operation and in the choice of thresholds. For the case of traditional string edit distance, some methods have been proposed for the basic edit operations—insertion, deletion, and substitution [5, 17]. Parametric video edit distance operations can borrow ideas from these proposals for their realization, with special attention to the new edit operations.

One major problem with the vstring approach is the high dependence on the robustness of features used to derive the vstrings. For instance, the edit distance relies more on the underlying index features for spatial information in the frames. Moreover, if the index features are not invariant under certain changes in the video sequence, such as illumination or partial occlusion, edit distances that are computed based on the vstring representation may not be very reliable. The other problem is the computational time and space that may be needed, especially when the database and the query sequences are both long. Ideas from fast algorithms for traditional string patterns [22, 30] can be used to speed up the vstring distance computation. One can still identify other issues that deserve some attention in using the proposed vstring edit distance, such as multidimensional vstring edit distance and context-dependent edit cost functions.

APPENDIX A: EDIT COST FOR THE SWAP OPERATION

The swap operation can be realized by a sequence of insertion, deletion, or substitution operations. For the special operations to be useful, however, the cost should be less than the cost of using the traditional edit operations. Consider the schematic below; symbol block S_1 is to be swapped with symbol block S_2:



SWAP PROCEDURE. $\Delta = \min\{\Delta_d, \Delta_q\}$: If $\Delta = \Delta_d$, perform swap on A; if $\Delta = \Delta_q$, perform swap on B. Let $s_b =$ swap block size and $\Delta_{bs} =$ separation block size. That is, $s_b = |S_1| = |S_2|$ and $\Delta_{bs} = \Delta$. The cost of the block-swap α_{bswap} will be:

$$\begin{aligned}\alpha_{bswap} &= (s_b + \Delta_{bs})\alpha_{del} + (s_b + \Delta_{bs})\alpha_{ins} \\ &= (s_b + \Delta_{bs})(\alpha_{del} + \alpha_{ins}).\end{aligned}$$

If sequential swap is used—i.e., no block swap operations are allowed, the swap block size will be 1, since only single symbols are swapped. The sequential swap will be performed once for each symbol in the swap block—a total of s_b times. Let $\Delta_{ss} =$ size of separation block for sequential swap, $\alpha_{seqswap} =$ cost of sequential swap operation. Since we are still considering the same string and the same swap operands, we should have:

$$1 + \Delta_{ss} + 1 = |S_1| + \Delta_{bs} + |S_2|, \quad \text{or} \quad \Delta_{ss} = 2(s_b - 1) + \Delta_{bs};$$

that is,

$$\begin{aligned}\alpha_{seqswap} &= (1 + \Delta_{ss})(\alpha_{del} + \alpha_{ins})s_b \\ &= (2s_b + \Delta_{bs} - 1)(\alpha_{del} + \alpha_{ins})s_b.\end{aligned}$$

Note that s_b and Δ_{bs} refer to their respective values for the block-swap operation. Clearly $\alpha_{seqswap} \geq \alpha_{bswap}$. Equality holds only when s_b, the swap block size is 1, in which case any of the methods (block swap or sequential swap) can be used.

APPENDIX B: SYMBOLS LIST

A, B	Sequence of symbols, or video sequences represented as vstrings,
n	Number of symbols in A
m	Number of symbols in B
Σ	Symbol alphabet
a, b	Symbols in Σ
ε	Null symbol
S_E	Sequence of edit operations
$c(s)$	Cost of edit operation s
$c(S_E)$	Cost of using S_E
$S_{A\rightarrow B}$	Set of all edit sequences that transform A into B
$S_{A\rightarrow B\min}$	Sequence in $S_{A\rightarrow B}$ with minimum cost
$D(A, B)$	Edit distance between A and B
α	Cost function for the edit operations
f_v	Numerical value of an index feature
$q(f_v)$	Quantization level for f_v
O_p	Set of all edit operations
$\$$	Symbol for video scene break
Σ_i	Symbol value for the ith symbol in the alphabet Σ
α_p	Set of respective costs for each edit operation in O_p
$d({}_b^a O_l)$	Distance between symbols a and b using edit operation O_l
$d_s(A, B)$	Symbolic edit distance between A and B
$d_b(A, B)$	Base distance between A and B
$d_p({}_{b_i}^{a_i} O_i)$	Base distance between symbols, using the feature values
$\tilde{f}_v(X)$	Average symbol value for string X
$D_v(A, B)$	vString edit distance between A and B
β	Weighting function for d_s and d_b
A_f	Internal representation for A after fusion operation
k_p	k-Difference threshold for normalized vstring distances
η	Number of subframes for each video frame

REFERENCES

1. D. A. Adjeroh, I. King, and M. C. Lee, Video sequence similarity matching, in *Proceedings, IAPR International Workshop on Multimedia Information Analysis and Retrieval, Hong Kong, August 1998*, Lect. Notes in Comput. Sci., Vol. 1464, pp. 80–95, Springer-Verlag, New York/Berlin, 1998.
2. D. A. Adjeroh and M. C. Lee, Adaptive transform domain video scene analysis, in *Proceedings, IEEE Multimedia'97*, Ottawa Canada, June 3–6, 1997.
3. G. Ahanger and T. D. C. Little, A survey of technologies for parsing and indexing digital video, *J. Visual Commun. Image Rep.* **7**(1), 1996, 28–43.
4. A. Amir and G. Calinescu, Alphabet independent and dictionary scaled matching, in *LNCS-1075, Combinatorial Pattern Matching 1996*, pp. 320–334.
5. H. Bunke and J. Csirik, Parametric sting edit distance and its application to pattern recognition, *IEEE Trans. Systems Man Cybern.* **25**(1), 1995.
6. H. Bunke and A. Sanfeliu (Eds.), *Syntactic and Structural Pattern Recognition: Theory, and Applications*, World Scientific, Singapore, 1990.
7. C.-W. Chang and S.-Y. Lee, Video content representation, indexing, and matching in video information systems, *J. Visual Commun. Image Rep.* **8**(2), 1997, 107–120.
8. S. K. Chang and E. Jungert, *Symbolic Projection for Image Information Retrieval and Spatial Reasoning*, Academic Press, London, 1996.
9. S.-K. Chang, Q.-Y. Shi, and C.-W. Yan, Iconic indexing by 2D strings, *IEEE Trans. Pattern Anal. Mach. Intell.* **9**(3), 1987, 413–428.
10. W. I. Chang and J. Lampe, Theoretical and empirical analysis of approximate string matching algorithms, in *LNCS 644, Combinatorial Pattern Matching, 1992*, pp. 175–184.
11. R. J. Compesi and R. E. Sherrifs, *Video Field Production and Editing*, Allyn Bacon, Boston, 1997.
12. N. Dimitrova and F. Golshani, Motion recovery for video content classification, *ACM Trans. Inform. Systems* **13**(4), 1995, 408–439.
13. M. Flickner *et al*, Query by image and video content: The QBIC system, *IEEE Comput.* Sept. 1995, 23–31.
14. K.-S. Fu and T.-F. Fan, Tree translation and its application to a time-varying image analysis problem, *IEEE Trans. Systems Man Cybern.* **12**(6), 1982, 856–866.
15. K.-S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
16. R. Giancarlo and R. Grossi, Multi-dimensional pattern matching with dimensional wildcards: Data structures and optimal on-line search algorithms, *J. Algorithms* **24**, 1997, 223–265.
17. D. Gusfield, K. Balasubramanian, and D. Naor, Parametric optimization of sequence alignment, *Algorithmica* **12**, 1994, 312–326.
18. F. Idris and S. Panchanathan, Review of image and video indexing, *J. Visual Commun. Image Rep.* **8**(2), 1997, 146–166.
19. M. Iorka and Masato Kurokawa, Estimation of motion vectors and their application to scene retrieval, *Mach. Vision Appl.* **7**, 1994, 199–208.
20. D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6**(2), 1997, 323–350.
21. J.-S. Lee, D. K. Kim, K. Park, and Y. Cho, Efficient algorithms for approximate string matching with swaps, in *LNCS 1264, Combinatorial Pattern Matching, 1997*, pp. 28–39.
22. E. W. Myers, A sublinear algorithm for approximate keyword searching, *Algorithmica* **12**, 1994, 345–374.
23. A. Pentland, R. W. Picard, and S. Sclaroff, Photobook: Content-based manipulation of image databases, *Int. J. Comput. Vision* **18**(3), 1996, 233–354.
24. H. S. Sawhney and S. Ayer, Compact representation of videos through dominant and multiple motion estimation, *IEEE Trans. Pattern Anal. Mach. Intell.* **18**(8), 1996, 814–830.
25. R. R. Schultz and R. L. Stenvenson, Extraction of high-resolution frames from video sequences, *IEEE Trans. Image Process.* **5**(6), 1996, 996–1011.
26. P. H. Sellers, The theory of computation of evolutionary distances: Pattern Recognition, *J. Algorithms* **1**, 1980, 359–373.
27. L. Teodosio and W. Benser, Salient video stills: Content and context preserved, in *Proceedings, ACM Multimedia'93*, Anaheim, California, August, 1993, pp. 39–46.
28. M. G. Thomason and E. Granum, Dynamic programming inference of Markov networks from finite sets of sample strings, *IEEE Trans. Pattern Anal. Mach. Intell.* **8**(4), 1986, 491–501.

29. W.-S. Tsai and S.-S. Yu, Attributed string matching with merging for shape recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **7**(4), 1985, 453–463.
30. E. Ukkonen, Finding approximate patterns in strings, *J. Algorithms* **6**, 1985, 132–137.
31. A. Wagner and M. J. Fischer, The string-to-string correction problem, *J. Assoc. Comput. Mach.* **21**, 1974, 168–173.
32. F. Wendling, J. J. Bellanger, J. M. Badier, and J. L. Coatrieux, Extraction of spatio temporal signatures from depth EEG seizure signals based on objective matching in warped vectorial observations, *IEEE Trans. Biomed. Eng.* **43**(10), 1996, 990–1000.
33. N. Yazdani and Z. M. Ozsoyoglu, Sequence matching of images, in *Proceedings, 8th International Conference on Scientific and Statistical Database Management, 1996*, pp. 53–62.
34. H. J. Zhang, *et al*, An integrated system for content-based video retrieval and browsing, *Pattern Recognition* **30**(4), 1997, 643–658.