A Distance Measure for Video Sequence Similarity Matching

D. A. Adjeroh, M.C. Lee and I. King
[donald, mclee, king] @cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin N.T., Hong Kong

Abstract- *Contrary to current approaches which generally treat the video data as a random collection of static images, content-based video retrieval requires methods for video sequence-to-sequence matching incorporating the temporal order inherent in video data. In this paper, we formulate the problem of video sequence-to-sequence matching as a pattern matching problem. New string edit operations required for the special characteristics of video sequences and the unique features of the vstring representation are introduced. Based on the edit operations, the* vstring edit distance *is proposed as a new similarity measure for video sequence matching.*

Keywords: content based video retrieval; video sequence-to-sequence matching; vstrings; vstring edit distance; distance measures, multimedia databases.

1. Introduction

The temporal constraints that accompany most multimedia data types (such as video, audio and animation) are one of the unique characteristics of multimedia information. The inherent temporal ordering of the data and the computational burden due to the huge data volumes however pose major problems for content-based retrieval. Current methods avoid the problem by performing the retrieval based on some surrogate of the original sequence (such as compact representations [Sawhney96], or key frames). They generally consider the surrogates as an image, and then use image matching methods to compare the scenes [Flickner95, Pentland96]. The problem with this simple image-based approach is the loss of the temporal information inherent in a video sequence: though the surrogates could be generated based on some motion and temporal information in the video, typically, such information are lost during the generation process. The deficiencies of simple image-based methods have been recognised, and recently other approaches have been proposed [Chang97, Dimitrova95, Ioka94, Zhang97].

The importance of the temporal constraints have led to recent efforts to incorporate them in video retrieval, by treating the video data in its natural form – as an *ordered* sequence of frames [Adjeroh98, Yazdani96]. Adjeroh, King, and Lee [Adjeroh98] have described the general problems in video sequence similarity matching, and proposed the *vstring (*video *string*) as an appropriate representation for the video sequence, when the objective is similarity matching. In this work, we use the vstring representation as a basis, and develop an edit distance based similarity measure for video sequence matching. Video retrieval techniques involving the temporal order in the video data will equally be of interest in other application areas, such as music and audio retrieval [Wold96], TV broadcasting, medical imaging, biomedical monitoring[Wendling96], crime investigation (copyright infringement), etc.

2 The Video Sequence Matching Problem

The video sequence similarity matching problem can be stated as follows: given two video sequences, determine how similar the two sequences are. Formally, given two video sequences: V_1, V_2, and some possible variation in the frames making up the sequences, we wish to find some function that will take V_1 and V_2 as its input, and return some (possibly quantitative) indicator of their similarity. Then we can compare the similarity indicator with a pre-stated (possibly user-defined) threshold of similarity, and declare V_1 and V_2 similar if they pass the threshold.

Unlike in the case of exact matching, since we are interested only in *similarity* between the sequences, the V_1 and V_2 need not be of the same length, and the size of the frames in each sequence need not be the same. Conceptually, V_1 can represent the entire database, while V_2 is just a short query sequence. Here, we would be interested in knowing if there exists any subsequence of V_1 that is similar to V_2. In video sequence matching, three basic types of matching can be identified: (i) *scene-to-scene*: check if two scenes are similar; (ii) *scene-to-sequence*: check if a scene similar to the query scene occurs in the database sequence; (iii) *sequence-to-sequence*: check if a sequence similar to a query sequence occurs in the database sequence. The query can contain more than one scene. To handle the trivial case of sequence-to-scene matching, we simply assume that the database sequence is the longer of the two sequences, though this has no effect on the actual matching process. We observe that (ii) is a generalisation of (i), and will make use of methods for (i). Similarly, (iii) is a generalisation of (ii) and its solution will depend on solutions to (ii). The basic problem thus is finding solutions to (i): the scene-to-scene matching problem.

0-8186-8676-6/98 $10.00

3 Pattern Matching and Edit Distances

Our approach is motivated primarily by techniques used for approximate (string) pattern matching. 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. A variant of the pattern matching problem is *approximate pattern matching* in which k-mismatches 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 exceed k. The distance between two strings is calculated using the string edit distance. Given two strings $A: a_1a_2...a_n$ and $B: b_1b_2...b_m$, over an alphabet Σ, and a set of allowed edit operations, the edit distance between A and B is the minimum number of edit operations needed to transform A into B.

3.1 Edit Operation

An edit operation, usually written as $(x \rightarrow y)$, is a pair $(x,y) \neq (\varepsilon,\varepsilon)$, of strings where $|x| \leq 1$ and $|y| \leq 1$. When we apply the edit operation $(x \rightarrow y)$ on an input string S_I to obtain an output string S_O, we say that S_I has been transformed into S_O, via the edit operation $(x \rightarrow y)$. That is, there exist some strings S_1 and S_2, such that $S_I = S_1xS_2$ and $S_O = S_1yS_2$. Three basic types of edit operations are used: *ins- insertion* of a symbol, $(\varepsilon \rightarrow a)$; *del - deletion* of a symbol, $(a \rightarrow \varepsilon)$; and *subs - substitution* of one symbol with another $(a \rightarrow b)$; (ε represents the zero-length empty symbol, and $x \rightarrow y$ indicates that x is transformed into y). To any given edit operation $(x \rightarrow y)$, a cost $c(x \rightarrow y)$ is attached. The value of the cost is determined by use of a weighting function.

3.2 Edit Sequences and Edit Distance

An edit sequence is simply 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: $S_E = s_1s_2...s_l$, where $s_i \in \{ins, del, subs\}$. The cost of a given edit sequence is determined by the cost of the individual edit operations making up the sequence:

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

Given two strings, A and B, there may be more than one edit sequence that transforms string A into string B. Let $S_{A \rightarrow B}$ represent the set of all edit sequences that transform A into B. The edit distance $D(A,B)$ is given by the edit sequence with the minimum cost, that is:

$$D(A,B) = \min\{c(S_E) \mid S_E \in S_{A \rightarrow B}\}$$

We can constrain the cost for each edit operation $c(x \rightarrow y)$ to be a distance metric: $c(x \rightarrow y) \geq 0$ - non-negative; $c(x \rightarrow y) = 0$ - reflectivity; $c(x \rightarrow y) = c(y \rightarrow x)$ - symmetry; $c(x \rightarrow y) \leq c(x \rightarrow z) + c(z \rightarrow y)$ - triangular inequality. With such as constraint, and since the edit distance will always select the path with the minimum cost, it is easy to prove the following theorem:

Theorem-1: *If $c(x \rightarrow y)$ is a metric, the edit distance $D(A,B)$ between strings A and B is also a metric:* $D(A,B) \geq 0$; $D(A,A) = 0$; $D(A,B) = D(B,A)$; $D(A,B) \leq D(A,C) + D(C,B)$

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 criteria for triangular inequality is often not met in most multimedia retrieval environments, and the cost need not be symmetric. Also, as will be seen later, depending on the method used to form the symbols in the string, the cost of an edit operation may also depend on the specific symbols involved.

The edit distance between $A: a_1a_2...a_n$ and $B: b_1b_2...b_m$ is usually determined using some recurrence equations [Sellers80, Wagner74]:

initialisations

$$d_{0,0} = 0;$$

$$d_{i,0} = d_{i-1,0} + \alpha_{del}(a_i); d_{0,j} = d_{0,j-1} + \alpha_{ins}(b_j);$$

main recurrence

$$d_{i,j} = \min\begin{cases} d_{i-1,j} + \alpha_{del}(a_i); \\ d_{i-1,j-1} + \alpha_{subs}(a_i, b_j); \\ d_{i,j-1} + \alpha_{ins}(b_j); \end{cases}$$

where α_{del}, α_{ins}, and α_{subs} are the respective cost of deletion, insertion, and substitution edit operations. The example below shows the edit distance between two strings, for two different cost functions.

Exaple-1: Edit distance between two sets of strings, using the cost function: $\alpha = [\alpha_{del}\ \alpha_{ins}\ \alpha_{sub}] = [1\ 1\ 1]$. With $\alpha = [1\ 1\ 2]$, the edit distance will be 6 and 5 respectively for the two string pairs.

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=[2,3,1,2], B=[1,2,3,1,3,1,3];
α = [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

With the edit distance, approximate pattern matching is performed by checking if there exists a substring A_s of A, such that the edit distance between A_s and B is less than k. Different algorithms have been proposed for the string edit distance problem, primarily based on dynamic programming [Seller80, Wagner74]. This typically requires an $O(mn)$ computational time, and various improvements have been proposed. See [Chang92] for a review on fast pattern matching.

4 The *vString* Distance

The vstring representation was proposed in [Adjeroh98] as an appropriate representation for video sequences, when the objective is similarity matching. After transcribing the sequence data into an appropriate representation, the actual matching using the selected representation will still have to be performed. W first describe the vstring representation which is the basis for the proposed vstring distance.

4.1 The *vString* Representation

One method with which the information in a video sequence can be captured is the string representation. Here, the video sequence as described by the sequence of feature values is transformed into a sequence of symbols, with the symbols in the string drawn from a defined alphabet. 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 represents a class, and the set of all symbols form the alphabet. Thus, the total number of symbols will depend on the number of classes used. We call such a representation of the video sequence a *video string*, or *vstring* for short. For multiple features, we may have different classifications, leading to multiple alphabets, with possibly different cardinality. Thus, in such a case, we will have multidimensional video strings, with the strings from each feature forming a dimension. Yazdani and Ozsoyoglu [Yazdani96] used the string approach to match image sequences, using sequence lengths and their moments as features. The longest common subsequence was used as a measure of the similarity between the sequences.

Usually, symbols in traditional text strings are taken as presence/absence symbols – that is, the symbols either appear in the string or they do not appear, and two different symbols are assumed not to have much in common. For instance, the symbols "*a*", and "*b*" in the English alphabet are assumed not to have much in common. For the vstring, when the symbols are taken from an alphabet obtained from the classification of real valued features, we 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 multi-valued symbols. 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 other types of classification (e.g. semantic classification of the sequences), the symbols can be treated as the traditional presence/absence symbols.

More importantly, the video string representation provides an intuitive method to model various characteristics and phenomena observed in the video sequences – such as repetitions, reverse, fast forward, video scene breaks, etc. Details of the vstring representation and examples of how the basic video scene transitions such as fast forward, slow motion, reverse, and partial reverse can be modelled by the vstring are given in [Adjeroh98].

4.2 Video String Edit Distance

Motivated by the traditional methods used in the video editing process - assemble and insert editing, and the edit distance based process of string pattern matching, we propose a method for matching video sequences. The technique we propose is suitable for both frame by frame sequence representation and comparison, and for shot-by-shot comparisons, and accounts for the special features and edit operations needed for the video sequence. We call the resulting similarity indicator the *vstring edit distance,* since it is based on the vstring representation

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 as 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 [Compesi97]. In addition to the possible temporal re-arrangements, our notion of video editing also includes possible changes in the actual *content* of the frames making up the shots or sequences.

It is then obvious that traditional string edit distances will be inadequate to cope with video strings. For instance, the above content requirement implies that the

symbols involved in video strings 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 each symbol in the alphabet. The value will typically depend on the classification method adopted and the particular features from which the vstring is derived. We will 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.

4.3 New *vString* Edit Operations

On account of the special video edit operations (such as those used to effect special effect transitions), or special video functionalities (such as fast forward/reverse), or the differences that may arise from other video operations (such as frame skipping), 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. Here a vstring edit operation, written as $(x \rightarrow y)$, is a pair $(x,y) \neq (\varepsilon,\varepsilon)$, of strings where $|x| \geq 0$ and $|y| \geq 0$. This implies that, rather than mere single symbols, *x, y* 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 *fusion* operator.

For the case of video strings, we extend the traditional edit distance by defining some new edit operations, namely: *swap*, *fusion/fission* and *break* operations. Generally, the vstring edit operation O_p can be represented as follows: $O_p = \begin{Bmatrix} a \\ b \end{Bmatrix} O \Big\}$, 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 $\aleph = \{\varepsilon,\$\}$, $\aleph \cap \Sigma = \varnothing$, Σ is the vstring alphabet, and Σ^* stands for any combination of symbols from 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 similarity mapping function between video strings.

Swap: interchange two symbols (or blocks of symbol) in one of the strings: *abcde→adcbe*, (transpose *b* and *d*), *abdc→cbda* (transpose *a* and *c*). In video, apart from the temporal positioning, the actual contents of the frames in the sequence are also important is assessing the similarity between sequences. Thus the transposition operation will be useful in handling the special transitions and possible partial occlusions that may occur in a video sequence, or the cases of partial reverse. 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 edit operation is characterised by the size of the strings to be swapped, and the number of symbols separating them in the database string, and in the query string. This is illustrated below:

A: $S_1\ \Delta_d\ S_2$; B: $S_2\ \Delta_q\ S_1$

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 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: (i) $\Delta - swap : \Delta_d = \Delta_q = \Delta$ - 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. (ii) $\Delta_d - swap$: $\Delta_d < \Delta_q$ - swap operation is applied to the database string. (iii) $\Delta_q - swap$: $\Delta_q < \Delta_d$ - swap operation is applied to the query string.

With the above formulation, we 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. The implications on cost will be discussed in Section 6.

Fusion/fission: fusion: merge a consecutive stream of the same symbol into a single symbol: *aaa→a*; fission: split a single symbol into a stream of symbols all of the same type: *a→aaa*. This will be 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*), and similarly, many consecutive symbols of the same type can be merged into a single symbol (*fusion*).

Let [*aa...a*] (*p* symbols) be represented as a^p. The fusion/fission operation then performs a simple transformation: $a^p \rightarrow a^t$, $t = f(p) = 1,2,\ldots$. Clearly, $f(p)$ determines the extent of the fusion/fision operation. Define $f(p) = \left\lceil \frac{p}{r} \right\rceil$, r > 0; For example, 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 lengths of the database and query sequences. Typically, $t \leq \left\lceil \frac{n}{m} \right\rceil$, since in the extreme case, the query string will be made up of *m* identical

symbols: i.e. $B = b_1 b_2 ... b_m = b^m$ or $b_i = b,\ i = 1,2,...,m$. The fusion/fission operation is thus a form of normalisation or scaling[1] on the original strings. We call r the scale of the fusion operation: i.e. if $r \geq 1$, each r or less consecutive symbols of the same type are *scaled down* to a single symbol; if $r < 1$, each single symbol is *scaled up* to $\lfloor \frac{1}{r} \rfloor$ symbols, all of the same type. By simply making $t > p$, (i.e. $r < 1$), we obtain the fission/split operation. In general however, $t < p$ - i.e. fusion operation, and in practice, we can accomplish both the fusion and fission operations by use of only the fusion operator, for instance by pre-scanning A and B. The fusion operation 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 repetitive use of the insert or delete operations.

Break: insert or delete 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 modelled as an ordinary concatenation, or by mere insertion and deletion operations.

4.4 vString Edit Distance

The basis for the vstring (symbolic) representation for video sequences is the feature values, which are real (continuous) numbers, representing numerical quantities like colour, motion, etc. The vstring on the other hand have discrete values, and depend on the original feature values. We call the continuous feature values the *base/primary representation*, and the vstring the *symbolic/secondary representation*. For a more precise appraisal of the distance between two 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 later as the secondary or symbolic edit distance.

The use of the symbolic and the base distances also provides a natural way of handling the fact that the video string could be multidimensional, and that symbols in a vstring could be multi-valued, rather than mere presence/absence symbols as is usually the case in traditional text strings. For a given dimension of the vstring, the value of each symbol can be obtained from the feature value used to derive the strings in that dimension. Methods for multidimensional pattern matching[Amir94, Giancarlo97] 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 for the base edit distance. As in traditional edit distances, appropriate weights will have to be assigned to each of the new operations.

4.4.1 vString Symbolic Edit Distance

Using the new operations, we can derive a general similarity measure (actually distance measure in this case) for video strings as follows: Let O_p be the set of edit operations: O_p={*insertion, deletion, substitution, swap, fusion, break*}, and 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: $S_E = \left\{ {}^{a_1}_{b_1}O_1, {}^{a_2}_{b_2}O_2, ..., {}^{a_l}_{b_l}O_l \right\}$, 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 i; a_i and b_i are the two symbols[2] involved in *i-th* edit operation. We can then determine the symbolic edit distance between A and B based on the cost of using this particular edit sequence:

$$c(S_E) = \sum_{O_p^k \in O_p} \left(\sum_{O_i = O_p^k} \alpha_p^k d({}^{a_i}_{b_i}O_i) \right)$$

where α_p^k ($\alpha_p^k \in \alpha_p$) is the cost of edit operation O_p^k, for instance $\alpha_p^k = \alpha_{ins}$ if O_p^k=*insertion*. $d({}^{a}_{b}O_l)$ is a distance function whose result depends on a, b, and the edit operation O_l. Since we will typically have different sequences of edit operations that can transform A into B, we will actually have a set of such edit sequences:

$$S_{A \rightarrow B} = \left\{ S_E^1, S_E^2, ..., S_E^q \right\}$$

where S_E^i is the *i-th* edit sequence that transforms A into B. The symbolic edit distance between A and B, $d_s(A,B)$ is then given by the least cost edit sequence:

$$d_s(A,B) = \min\{c(S_E) \mid S_E \in S_{A \rightarrow B}\}$$

The symbolic edit distance is an extension of the traditional edit distance, with consideration of the special vstring edit operations. It is however not an accurate representation of the distance between the video strings. A more accurate measure of the distance is obtained by combining the symbolic edit distance and the base edit distance.

[1] Though scaling as used here is related to the notion of *scaled matching* introduced by Amir *et al* [Amir96], our definition of scaling is different. Here, scaling is only within the symbols in a string, while in [Amir96], scaling was defined on the entire string i.e. for $A = a_1 a_2 ... a_n$, $A^s = a_1^s a_2^s ... a_n^s \neq (AA...A) = A$ concatenated s times. s is the scaling factor.

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

4.4.2 vString Base Edit Distance

Since we assume the results from the symbolic part will still produce a general idea (though not very accurate) indication of the distance, we can make use of the edit sequence used for computing symbolic edit distance in calculating the base edit distance. That is, computing the base edit distance need only be performed AFTER computation of the symbolic edit distance. This will thus be calculated only for the minimum cost edit sequence. This can easily be done using any of the general distance metrics, such as the city-block, Euclidean, or the general Minkowsky metric. The use of the minimum cost edit sequence also implies that computation of the base edit distance will not add to the complexity of the symbolic edit distance, since at worst, only O(m) additional computation will be required.

Let $S_{A\to B_{\min}}$ represent the minimum cost edit sequence that transforms A into B. That is, $S_{A\to B_{\min}}$ is an ordered sequence of edit operations: $S_{A\to B_{\min}} = \left\{ {}^{a_1}_{b_1}\mathrm{O}_1, {}^{a_2}_{b_2}\mathrm{O}_2, \ldots, {}^{a_l}_{b_l}\mathrm{O}_l \right\}$ used to compute the symbolic edit distance. We define the base distance between A and B using the above minimum cost edit sequence:

$$d_b(A,B) = \sum_{i=1}^{l} d_b\left({}^{a_i}_{b_i}\mathrm{O}_i\right)$$

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

$$d_b(A,B) = \left[\sum_{i=1}^{l} \left[d_b\left({}^{a_i}_{b_i}\mathrm{O}_i\right) \right]^p \right]^{\frac{1}{p}}.$$

The base distance is thus dependent on the specific edit operation and also on the symbols involved in the operation. $d_b(.)$ is defined as follows for each of the vstring edit operations:

$$d_b\left({}^{a}_{b}\mathrm{O}\right) = \begin{cases} K_I + \left| f_v(b) - \tilde{f}_v(B) \right| & ;\textit{insertion} \\ K_D + \left| f_v(a) - \tilde{f}_v(A) \right| & ;\textit{deletion} \\ \left| f_v(a) - f_v(b) \right| & ;\textit{substitution} \\ \left| f_v(a) - f_v(b) \right| & ;\textit{swap} \\ K_F + \left| f_v(a) - f_v(b) \right| & ;\textit{fusion} \\ K_B & ;\textit{break} \end{cases}$$

where K_B, K_D, K_F, and K_I are constants which could be application dependent, $f_v(a)$ is the normalised feature value of symbol a, and $\tilde{f}_v(X)$ is the average feature value for the string X. Let a and b above be symbol-blocks, rather than single symbols: i.e. $a = a_1 a_2 \ldots a_s$ $b = b_1 b_2 \ldots b_s$. Then, we define the base distance for the block-swap edit operation as follows:

$$d_b\left({}^{a}_{b}\mathrm{O}\right) = \sum_{i=1}^{s} \left| f_v(a_i) - f_v(b_i) \right| \; ;\textit{block} - \textit{swap}$$

We have tried to make the above definition for the base distance general. The constants can be assigned the value of zero where desired, for instance, for the fusion operation. Clearly,

$$K_B \geq \min\{\forall(a,b) \; \left| f_v(a) - f_v(b) \right| + \max\{\mathrm{K_I}, \mathrm{K_D}\}$$

since it should cost more to insert/delete a scene break to insert/delete an ordinary symbol.

4.4.3 *vString* Edit Distance

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

$$D_v(A,B) = \beta . d_b(A,B) + (1-\beta) . d_s(A,B)$$

where β is a weighting function ($0 \leq \beta \leq 1$) which biases the weight to either the symbolic or the base distances, d_s and d_b are suitably normalised to within the same range, e.g. d_b can easily be normalised to fall within [0 1], while d_s can be normalised by dividing with the query string length.

The symbolic distance helps to reduce differences due to noise variations, and problems of invariance in the specific feature used to form the strings. The base distance on the other hand uses the actual feature values, and is thus more accurate - thereby reducing the problem caused by the inherent loss of accuracy in the vstring representation. The appropriate choice of the weighting function is still an issue, but can generally be application dependent. The vstring edit distance described above may or may not be a metric. However, the following theorem can be proved:

Theorem-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 (represented by their vstring) is also a metric:* *i.e.* $D_v(A,B) \geq 0$; $D_v(A,A) = 0$; $D_v(A,B) = D(B,A)$; $D_v(A,B) \leq D_v(A,C) + D_v(C,B)$

5. Preliminary Experimental Results

We tested the proposed edit distance based measure on some test video sequences. We present the results for only one sequence. The sequence is taken from a news video. Space constraints does not permit us to include the video frames in this paper. So we describe (and group) the video contents in general terms:

- $v_1, v_2, v_3, v_7, v_{10}$: anchor man + (inset) North Korean flag, South Korean flag, Boris Yeltsin, tennis, respectively;
- v_4, v_5, v_6: large hall with seats (different views);
- v_8, v_9: air plane (blue sky, green grass background respectively);
- v_{11}, v_{12}: tennis game (green lawn, different views); v_i represents the *i-th* video scene.

The distance between the scenes obtained using the vstring distance is shown in the table below. The

results are for an alphabet size $|\Sigma|=4$, with $K_D = K_I = \frac{1}{|\Sigma|-1}$, cost function [1 1 1], β=0.5. Each frame in the sequence was divided into 4 subframes, and the vstring was formed using the colour mean and standard deviation form the subframes. The base edit distance was calculated using the symbol values (not the exact feature values - see section on discussion).

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
length	155	386	563	53	85	38	121	46	110	414	434	133
v_1	0	0.17	0.17	0.55	0.53	0.54	0.04	0.62	0.50	0.16	0.56	0.56
v_2		0	0.07	0.59	0.59	0.61	0.20	0.90	0.56	0.03	0.51	0.61
v_3			0	0.68	0.65	0.72	0.20	1.00	0.58	0.08	0.52	0.62
v_4				0	0.11	0.09	0.55	0.42	0.25	0.58	0.28	0.26
v_5					0	0.16	0.54	0.44	0.25	0.58	0.28	0.25
v_6						0	0.55	0.43	0.29	0.62	0.30	0.29
v_7							0	0.59	0.50	0.18	0.58	0.57
v_8								0	0.31	0.86	0.49	0.46
v_9									0	0.53	0.32	0.30
v_{10}										0	0.49	0.58
v_{11}											0	0.13

Table-3: Normalised vstring distance test video.

The table shows that, in general the scenes that belong to the same group recorded smaller distances, when compared with those belonging to another group, implying that the proposed method provides some reasonable measure of the similarity between the video scenes. It may also be observed that the effect of scene lengths have largely been reduced, by length normalisation. The results are only preliminary. Various issues are still under investigation, such as the effect of alphabet size, the no of subframes, appropriate cost functions and weights for d_s and d_b.

6. Discussion

6.1 Edit Cost for the Special Edit Operations

We can access the cost of the special vstring edit operations by considering the basic edit operations from which they are derived. For instance, the swap operation can be realised by a sequence of insertion and deletion operations, or by use of substitution operations. For the special operations to be useful however, the cost should be less than the corresponding cost of using the traditional edit operations. We give a genaral description of the cost for the *block-swap* operation. When the swap blocks are of size 1, we have the ordinary swap operation.

Example-2: Using the *swap* edit operation:
Given: *A: cde efx abc* and *B: abc efx cde*
Swap symbol blocks *cde* and *abc*

Using *sequential swap* (3 swap operations)
swap *a* and *c*: *A: cde efx abc* *B: cbc efx ade*
swap *b* and *d*: *A: cde efx abc* *B: cdc efx abe*
swap *c* and *e*: *A: cde efx abc* *B: cde efx abc*

Using *block swap* (1 single block-swap operation)
swap [*abc*] and [*cde*]:
A: cde efx abc B: cde efx abc

Swap Cost (for brevity, we skip the details)
Let α_{bswap} = cost of block-swap operation, $\alpha_{seqswap}$ = cost of sequential swap, s_b =swap block size, ($s_b = |S_1| = |S_2|$), Δ_{bs} =size of separation between swap blocks, and $\Delta_{bs} = \min\{\Delta_d, \Delta_s\}$. The cost of the block swap operation will be:

$$\alpha_{bswap} = (s_b + \Delta_{bs})(\alpha_{del} + \alpha_{ins})$$

If sequential swap is used - i.e. no block swap operations are allowed, the cost will be:

$$\alpha_{seqswap} = (2s_b + \Delta_{bs} - 1)(\alpha_{del} + \alpha_{ins})s_b$$

It is clear from the above that $\alpha_{seqswap} \geq \alpha_{bswap}$. Equality holds only for s_b =1, in which case any of the methods can be used. We can do a similar analysis for the case of the fusion edit operation. The overall cost will depend once again on the costs of the insertion and deletion operations, since these are the elementary operations from which the fusion operation is realised. Another factor in determining the cost will be the scale parameter *r*; the larger the value of *r*, the more the cost.

6.2 Weighted *vString* Edit Distance

The distance between any two strings 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 is to be allowed in the match. As one may have noted from the previous sections, a uniform cost for each of the edit operations may not accurately model the importance of each operation in the video sequence. For instance, a deletion operation (which can be seen as 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. Also, a consideration of the possibly multi-valued nature of the symbols in a video string implies that the edit distances (even when the same weight is used) may no longer be integer values.

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, delete, and substitution [Bunke95, Gusfield94, Rice97]. Parametric vstring edit distance operations can borrow ideas from these proposals for their realisation. Attention will have to be

paid to the special edit operations, and the distance normalisation for a given cost function.

Apart from the edit costs, appropriate methods for defining the weights between the base edit distance and the symbolic edit distance are also required. One more issue is the assumption that the base distance should be based on the minimum cost path, which is used to determine the symbolic distance, d_s. This need not be the case. In fact d_b can be computed separately - the method used for the experiments. Alternatively, it can be calculated for each edit operation, as part of the decision on the minimum cost edit sequence. That is, at each edit step, the vstring edit distance computation will be performed for d_s and d_b for each edit operation, and the combined minimum edit cost determined BEFORE proceeding to the next edit step. The result will be more accurate, but at an additional computational cost. Also, new weights may be required for d_s and d_b if this approach is adopted.

A simplification of the base distance can be obtained by using the serial number of the symbols as the symbol value, and then replacing the feature values in the computation of d_b with the symbol values. Let Σ_i = *i-th* symbol in the alphabet Σ. Then, its symbol value will be: $f_v(\Sigma_i) = i; i = 1,2,...,|\Sigma|$.

One major problem with the vstring approach is the high dependence on the robustness of features used to derive the vstrings. For instance, if the initial features are not invariant under certain changes in the video sequence, such as illumination or partial occlusion, edit distances 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. One can still identify other issues that need some attention in using the proposed vstring edit distance, such as multidimensional vstring edit distance, context dependent edit cost functions, normalisation for the vstrings, and also for the edit distances, etc.

References

[Adjeroh98] Adjeroh D.A., King, I., and Lee, M.C, "Video sequence similarity matching", to appear, *Proc., MINAR'98,* Hong Kong, August 1998.

[Amir96] Amir A., and Calinescu G., "Alphabet independent and dictionary scaled matching", *LNCS*-1075, *Combinatorial Pattern Matching*, 320-334, 1996.

[Amir94] Amir A., Benson G., and Farach M., An alphabet independent approach to two dimensional pattern matching", *SIAM J. Comput.*, 23, 2, 313-323, 1994.

[Bunke95] Bunke H. and Csirik J., "Parametric sting edit distance and its application to pattern recognition", *IEEE Trans. Sys., Man & Cybernetics*, 25, 1, 1995.

[Chang97] Chang C-W, and Lee S-Y, "Video content representation, indexing, and matching in video information systems", *J. Visual Communication and Image Representation*, 8, 2, 107-120, 1997.

[Chang92] Chang W.I. and Lampe J. "Theoretical and empirical analysis of approximate string matching algorithms", *LNCS 644, Combinatorial Pattern Matching*, pp. 175-184, 1992

[Dimitrova95] Dimitrova N. and Golshani F., "Motion recovery for video content classification", *ACM Trans. Info. Sys.*, 13, 4, 408-439, 1995.

[Flickner95] Flickner M. *et al*, "Query by image and video content: the QBIC system", *IEEE Computer*, 23-31, Sept. 1995.

[Giancarlo97] Giancarlo R. and Grossi R., "Multi-dimensional pattern matching with dimensional wildcards" data structures and optimal on-line search algorithms", *Journal of Algorithms*, 24, 223-265, 1997.

[Gusfield94] Gusfield D., Balasubramanian K., and Naor D., "Parametric optimization of sequence alignment", *Algorithmica*, 12:312-326, 1994.

[Iorka94] Iorka M., and Masato Kurokawa, "Estimation of motion vectors and their application to scene retrieval", *Machine Vision and Applications*, 7:199-208. 1994.

[Pentland96] Pentland A., Picard R.W., and Sclaroff S., "Photobook: content-based manipulation of image databases", *International Journal of Computer Vision*, 18, 3, 233-354, 1996.

[Rice97] Rice S.V, Bunke H., and Nartker T.A., "Classes of cost functions for string edit distance", *Algorithmica*, 18: 271-280, 1997.

[Sawhney96] Sawhney H.S., and Ayer S., "Compact representation of videos through dominant and multiple motion estimation", *IEEE Trans. PAMI* 18,8,814-830, 1996.

[Sellers80] Sellers P.H, "The theroy of computation of evolutionary distances: pattern ecognition", *Journal of Algorithms*, 1, 359-373, 1980.

[Wagner74] Wagner A. and Fischer M.J, "The string-to-string correction problem", *J. ACM*, 21: 168-173, 1974.

[Wendling96] Wendling F., Bellanger J.J, Badier J.M, and Coatrieux J.L, "Extraction of spatio temporal signatures from depth EEG seizure signals based on objective matching in warped vectorial observations", *IEEE Trans. Biomedical Engineering*, 43, 10, 990-1000, 1996.

[Wold96] W. Wold *et al*, "Content-based classification, search and retrieval of audio", *IEEE Multimedia*, 3, 3, 27-36, 1996.

[Yazdani96] Yazdani N and Ozsoyoglu Z.M., "Sequence matching of image", *Proc., 8th Int'l Conf. on Scientific and Statistical Database Management*, pp. 53-62, 1996.

[Zhang97] Zhang H.J, et al, "An integrated system for content-based video retrieval and browsing", *Pattern Recognition*, 30,4,643-658, 1997.