Two–dimensional string notation for representing video sequences

Kim Shearer, Dorota Kieronska, Svetha Venkatesh

Department of Computer Science
Curtin University of Technology

ABSTRACT

Most current work on video indexing concentrates on queries which operate over high level semantic information which must be entirely composed and entered manually. We propose an indexing system which is based on spatial information about key objects in a scene. These key objects may be detected automatically, with manual supervision, and tracked through a sequence using one of a number of recently developed techniques. This representation is highly compact and allows rapid resolution of queries specified by iconic example.

A number of systems have been produced which use 2D string notations to index digital image libraries. Just as 2D strings provide a compact and tractable indexing notation for digital pictures, a sequence of 2D strings might provide an index for a video or image sequence. To improve further upon this we reduce the representation to the 2D string pair representing the initial frame, and a sequence of edits to these strings. This takes advantage of the continuity between frames to further reduce the size of the notation.

By representing video sequences using string edits, a notation has been developed which is compact, and allows querying on the spatial relationships of objects to be performed without rebuilding the majority of the scene. Calculating ranks of objects directly from the edit sequence allows matching with minimal calculation, thus greatly reducing search time. This paper presents the edit sequence notation and algorithms for evaluating queries over image sequences. A number of optimisations which represent a considerable saving in search time is demonstrated in the paper.

Keywords: video indexing, 2D strings, spatial reasoning,

1 INTRODUCTION

The accepted method of locating items of interest in large data collections is indexing. Whether the data is stored on paper or in a computer, a well constructed index is invaluable in increasing the efficiency of data retrieval. Techniques from computer vision and spatial reasoning have been successfully applied to indexing libraries of photographic digital pictures,[7,13] however the increased capacity of computers to store and process information has opened new areas of study. The ability to play video sequences on a computer has led to growth in the use of computers for video processing and the birth of multimedia. In this paper we discuss indexing techniques for image sequences. We make no distinction in this paper between video data and image sequences, such as GIS data or medical imaging, and refer to both either as video or sequences.

Digital storage of video requires large amounts of space, and can be very slow to process. Rather than process the video data directly, we would prefer to produce an index which allows filtering before accessing the full data.

0-8194-1970-2/95/$6.00

Current indexing techniques for video are generally either low level, providing locations of camera actions and edit positions, or high level semantic descriptions. As video is such an information rich medium there may be many valuable indexing schemes. An ideal video database system should provide a number of complementary indexing systems which are intergated to present a coherent query system. We propose extending spatial reasoning techniques used in photographic databases for use with video, as a complement to existing methods.

The two common forms of low level video information are camera motion and edit or cut detection. The location of cuts within video sequences may be accomplished by a number of methods,[12,1] all of which are highly reliable and automated. A cut is the place in a video where one shot ends and another begins. A shot is a sequence of frames taken by one camera in a continuous shoot. The detection of cuts allows a long video or image sequence to be broken into smaller units, which are more likely to be homogeneous in characteristics.

Camera motion detection may also be done automatically with high reliability. By camera motion we mean the basic motions available to the camera, such as: *booming*, *dollying* or *tilting*. These camera motions may be of direct interest to the query system, as camera motion can be used to identify scenes which are otherwise very similar.

The high level techniques used in video indexing are based on semantic description.[4,8] These techniques require a manually created description of each shot in the database. Shots may then be accessed using any data which is included in the semantic descriptions. This may be the ideal index for some applications, however there are major difficulties. It takes a great deal of manual labour to produce the semantic descriptions for a database of shots, and once the indices are created there may be serious consistency problems. Each individual will draw on their own experiences and view point to produce an index, and may do so with differing goals. Creating a consistent standard could in itself be difficult. This is likely to become increasingly problematic as libraries become more widely available over networks.

Spatial indexing of videos provides an index which is based on the image contents, unlike low level indices, but cannot be generated completely automatically. The information on image content is restricted to the spatial relationships between objects, so there is no semantic interpretation, however creation of the index is not as labour intensive as the creation of semantic descriptions. It is also considerably simpler to ensure consistency for spatial indices, by providing attribute lists, from which object attributes are chosen.

There are a variety of systems for representing and reasoning about spatial relationships. All are aimed at compactness of representation and efficient querying. The system used as the basis for the work in this paper is 2D B–strings.

2 2D strings

The early work on 2D strings was presented in 1987 by Chang, Shi and Yan,[6] as part of a project to produce an iconic indexing system for pictorial databases. Pictures are converted to a pair of strings, called 2D strings, which are stored in a database. Queries are expressed in the same 2D string notation, allowing the database to be searched using a simple matching scheme. Matching strings which are detected in the database are decoded into simple iconic representations of the original picture, which may be displayed as a response to the query.

The alphabet of 2D strings is a set V of object labels and a set A containing the operators $\{=, <, :\}$.

DEFINITION 1. *A 1D string, defined over a set S, is any string $x_1, x_2, \ldots x_n$ where $n \geq 0$ and $\forall j : x_j \in S$.*

DEFINITION 2. *A 2D string over V is defined to be*

$$(x_1, y_1, x_2, y_2, \ldots \quad y_{n-1}, x_n, \quad x_{p(1)}, z_1, x_{p(2)}, \ldots z_{n-1}, x_{p(n)})$$

where	
$x_1 \ldots x_n$	*is a 1D string over* V
$p : \{1..n\} \rightarrow \{1..n\}$	*is a permutation function over* $\{1 \ldots n\}$
$y_1 \ldots y_{n-1}$	*is a 1D string over* A
$z_1 \ldots z_{n-1}$	*is a 1D string over* A

The function p is a permutation which maps a symbol from the first string of a 2D string pair to its corresponding symbol in the second string.

The first string in a 2D string pair expresses object relations along the u axis, by convention this is the horizontal axis, increasing left to right. The second string expresses relations along the v axis, the vertical axis increasing bottom to top. The operators '=' and '<' have the usual meanings of *equal* and *less than*, with *less than* being to the left of for the u axis, and below for the v axis. The operator ':' means that two objects are in the same object set.

The matching algorithm for 2D strings uses a simple ranking scheme for the object symbols in strings. A symbol in a string has a rank defined to be one plus the number of '<' operators preceding the symbol. Using this ranking scheme Chang *et al* define three different types of matching, providing three levels of strictness of match, with type-2 being an exact match and type-1 and type-0 being progressively less strict. The three types of match have the following definition:
A string S is a type-i subsequence of a string T if

1. S is contained in T
2. If $a_1 w_1 b_1$ is a substring of S, $a_2 w_2 b_2$ is a substring of T, where a_1 matches a_2 and b_1 matches b_2 then

   Type–0 $rank(b_2) - rank(a_2) \geq rank(b_1) - rank(a_1)$ or $rank(b_1) - rank(a_1) = 0$

   Type–1 $rank(b_2) - rank(a_2) \geq rank(b_1) - rank(a_1) \geq 0$ or $rank(b_2) - rank(a_2) = rank(b_1) - rank(a_1)$

   Type–2 $rank(b_2) - rank(a_2) = rank(b_1) - rank(a_1)$

This representation suffers from a major inadequacy, displayed in figure 1. In this figure there are three pictures which, using 2D strings, all result in the same representation: $(A = B < C, A < B = C)$. This is due to 2D strings being a point based representation, with object position being determined by the object's centroid. 2D strings discard extent information, so that the obvious differences between the pictures in figure 1 are not retained in the 2D strings.

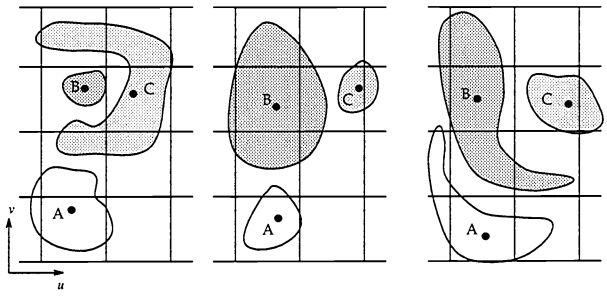

Figure 1: Ambiguous pictures using a 2D string representation

Jungert proposes E–strings,[9] for extended 2D strings by adding operators which form the basis for much of the further work. The extra operators are derived from work in temporal reasoning by Allen.[2] Using the extra

operators allows the accurate expression of a much greater range of relationships. Unfortunately, as the *overlap* operator is not transitive, it is not possible to store the binary relationships in a unified structure using E–strings.

Later work uses an approach based on segmentation of objects into simple regions. A cutting mechanism is introduced making it possible to avoid the break of transitivity caused by the *overlap* operators. This approach was first suggested by Chang, Jungert and Li with G–strings.[5] In this system each object is cut against the minima and maxima of all other objects in the scene. Using the subobjects defined by the cuts, it becomes possible to express spatial relationships without using an *overlaps* operator. This allows the expression of pictures as two 1D strings, each expressing the relationships along an axis. The main drawback of this approach is that pictures with many objects can cause a very high level of segmentation. The number of segments increases rapidly as the number of objects grows, this reduces the efficiency of storage and greatly increases the time required to match pictures. The C–string system of Lee and Hsu[10] addresses the cutting problem directly by reinstating the *begin* and *end* operators (=| and |=), which had been omitted from G–strings. This allows a sharp reduction in the number of cuts required.

The characteristic of Chang's original system which was lost in later systems is that, being point based, all symbols are simply comparable. That is, all symbols are either equal, less than or greater than each other symbol. The cost of this simplicity is the loss of extent information, leading to greatly reduced discriminatory powers. A system which provides many of the features of Chang's point based system and the later interval based systems, is 2D B–strings.[11] B–strings represent objects by two symbols, one representing the start point along an axis, and another representing the end point along the axis.

The advantage of this representation is that all symbols in the strings are simply comparable, unlike the segmented objects of interval based systems, yet extent information is represented. The cost of each object requiring two symbols in each string is partially balanced by the reduced number of operators required, and the fact that objects are not segmented. Matching for B–strings uses three types, similar to Chang *et al.* These types are calculated using symbol ranks, and are based on relationship categories (due to Allen[2]) and orthogonal relationships. Symbols in B–strings are assigned a rank using the following equation

$$\text{rank}(s_i) = \sum_{j=1}^{i} f(s_j) \quad \text{where} \quad f(s) = \begin{cases} 1 & \text{if } s \neq \text{`='} \\ -1 & \text{if } s = \text{`='} \end{cases}$$

Type–0 matching is based on the five relationship categories. Taking each of the 13 spatial relationships along each axis, there are a total of 169 possible 2D spatial relations. These are categorised into five partitions, *disjoint*, *meet*, *contain*, *inside* and *partly overlap*. The partitions are defined by:

DEFINITION 3. *The relationship category of the spatial relationship between two objects a and b is:*

1. *Disjoint : if* end_rank*(a)*<begin_rank*(b)* $\vee$ end_rank*(b)* < begin_rank*(a) along either axis*
2. *Meet : if* end_rank*(a)*=begin_rank*(b)* $\vee$ end_rank*(b)*=begin_rank*(a) along one axis and the condition for disjoint is not satisfied on the other axis*
3. *Contain :* begin_rank*(a)*$\leq$begin_rank*(b)* $\wedge$ end_rank*(b)*$\leq$end_rank*(a) along both axes*
4. *Inside :* begin_rank*(b)*$\leq$begin_rank*(a)* $\wedge$ end_rank*(a)*$\leq$end_rank*(b) along both axes*
5. *Partly overlap : if no other relationship category holds*

Two objects which are a type–0 match between two pictures must have the same overlap properties, but the match is invariant under rotation and reflection. This is the least strict of the three matching types.

Type–1 matching uses orthogonal relationships between objects. The orthogonal relationship between two objects expresses the direction in which one object lies from the viewpoint of the other object. Orthogonal relationships are defined as follows:

DEFINITION 4. *Object a is said to be to the* east*(*west*) of another object b iff there is some part of a to the* east*(*west*) of the whole of b.*

DEFINITION 5. *Object a is said to be to the* north*(*south*) of another object b iff there is some part of a to the* north*(*south*) of the whole of b.*

We use the notation $aC_{ab}b$ to denote the relationship category for two objects a and b, $aO_{ab}b$ to denote orthogonal relationships and $aR_{ab}B$ to denote the exact spatial relationships between the objects. B–string matching may then be defined as follows:

DEFINITION 6. *Picture f' is a type–i subpicture of f if*

1. *All objects in f' are also in f*
2. *For any two objects, $a, b \in f'$*

   $aC_{ab}b$, $aO_{ab}b$ and $aR_{ab}b$ are in f and $aC_{ab}'b$, $aO_{ab}'b$, and $aR_{ab}'b$ and in f'

   $C_{ab}' = C_{ab} \Rightarrow$ Type–0 match

   $C_{ab}' = C_{ab} \wedge O_{ab}' = O_{ab} \Rightarrow$ Type–1 match

   $C_{ab}' = C_{ab} \wedge O_{ab}' = O_{ab} \wedge R_{ab}' = R_{ab} \Rightarrow$ Type–2 match

As noted above, type–0 matching is rotation and reflection invariant, type–1 matching uses orthogonal relationships to restrict matches to objects which are of similar orientation. Type–2 matching is the most strict, requiring that two objects be a type–1 match, and have the same spatial relationship along each axis.

3 B–STRING NOTATION FOR VIDEO SEQUENCES

The use of 2D strings to index video was first suggested by Arndt and Chang.[3] Their representation consists of an encoding of the initial frame as a number of sets of objects, and a sequence of edits to these sets which produce a representation for subsequent frames. The edits are interspersed with frame numbers, indicating the frame which the next sequence of edits will represent. The sets are derived from the Chang 2D string notation, with all objects having the same rank along an axis being placed in the same set. Further, the sets formed from the v axis are concatenated onto the end of the sets produced from the u axis. Consider the 2D u string $dog < cat\ mouse$ where the set $\{dog\}$ contains all objects of rank one, and $\{cat\ mouse\}$ contains all objects of rank two. The corresponding v string: $cat\ dog < mouse$ has sets; $\{cat\ dog\}$ which contains objects of rank one, and set $\{mouse\}$ of rank two. Thus the 2D string:

$$dog < cat\ mouse, cat\ dog < mouse$$

is encoded as the following list of sets:

$$\{dog\}\{cat\ mouse\}\{cat\ dog\}\{mouse\}$$

Each set is identified within the edit notation by an integer, where the sets are numbered from one. In the example given set two would be $\{cat\ mouse\}$, and set four $\{mouse\}$.

Let $o_1 \ldots o_n$ be the objects in a sequence, and $l_1 \ldots l_m$ the sets in the 2D set notation. The following operators are defined by Arndt and Chang:

$\oplus o_i l_j$ add the object o_i to the object set l_j

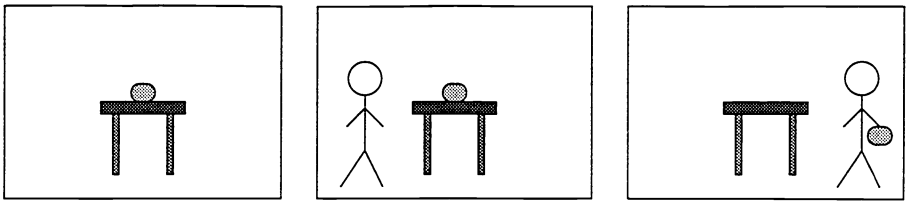

Figure 2: Three frame video sequence

$\ominus o_i l_j$	delete object o_i from the object set l_j
$\lhd\rhd\ l_i$	merge the sets l_i and l_{i+1} into one set, which becomes l_i
$\lhd\rhd\ l_i o_{i_1} \ldots o_{i_m}$	split set l_i, the objects $o_{i_1} \ldots o_{i_m}$ remain in set l_i, and the objects $o_j : o_j \in l_i \wedge o_j \notin \{o_{i_1}, \ldots, o_{i_m}\}$ form a new set l_{i+1}

The merge operator has the additional effect that merging l_i and l_{i+1} causes all sets l_j where $j > i$ to become l_{j-1}. Similarly for the split operation, when l_i is split to become l_i and l_{i+1}, all previous sets l_j where $j > i$ become l_{j+1}. The operators $\oplus$ and $\ominus$ also have the additional function of adding and deleting whole sets, the syntax for these functions is as follows:

$\oplus l_i$	create new set number i
$\ominus l_i$	delete the set number i

Thus there are two stages in the conversion, the conversion from 2D strings to sets, and from sets to the edit notation. To illustrate this conversion process we provide the following example, based upon the three frame sequence in figure 2.

$$\begin{array}{c} tablebox, tablebox \\ man < tablebox, mantable < box \\ table < manbox, mantablebox \end{array}$$

which corresponds to the following sequence of sets:

Frame 1	$\{table\ \ box\}$	$\{table\}\{box\}$
Frame 2	$\{man\}\{table\ \ box\}$	$\{man\ \ table\}\{box\}$
Frame 3	$\{table\}\{man\ \ box\}$	$\{man\ \ table\ \ box\}$

these sets would be converted to the compressed notation as:

1	$\{table\ \ box\}\{table\}\{box\}$	*Initial string*
2	$\oplus 1\ \ man\ 1\ man\ \ 3$	*Create set 1 and add man, add man to set 3*
3	$\ominus 1$	*Delete set 1*
	$\lhd\rhd\ 1\ table$	*Split set 1*
	$\oplus\ man\ 2$	*Add man to set 2*
	$\lhd\rhd\ 3$	*Merge sets 3 and 4*

There are two algorithms provided by Arndt and Chang for use with their representation. The first algorithm converts a sequence of 2D strings, each representing a video frame, to the set notation, the second algorithm

reverses this conversion. When encoding video, frames are first translated into their 2D string representation, then the sequence of strings is converted to *set edit* notation. Queries require first the conversion of the set notation back into a sequence of 2D strings, followed by standard 2D string matching. Arndt–Chang notation gives us a method for temporarily storing a video index in a temporary compressed format, to be expanded before use.

The system proposed in this paper offers not only an improved representation, but also a query system that does not require expansion and parsing of a sequence of 2D strings. Our system uses B–strings which, unlike Chang's 2D strings, incorporate extent information. This increases the descriptive power of the representation. The major improvement is the query process, which does not require the expansion and evaluation of a sequence of 2D strings. Spatial relationship changes are read directly from the edit sequence. This makes it simple to discard edits which do not affect the objects in the query, which reduces the processing required during querying.

Let $\mathcal{Q} = q_1 \dots q_m$ denote the query string and $\mathcal{R} = r_1 \dots r_n$ the reference string. Then let q_f denote the first symbol of $\mathcal{Q}$ to occurring $\mathcal{R}$, and let q_l denote the last symbol of $\mathcal{Q}$ to occurring $\mathcal{R}$. Then $\mathcal{R}$ can be expressed as $[r_1 \dots r_{f-1}][q_f \dots q_l][r_{l+1} \dots r_n]$. The first or last sets may be empty, and $q_f \dots q_l$ will usually contain elements not in $\mathcal{Q}$. There are no query symbols in either the first or third sets, therefore changes to these sets do not affect the spatial relationships of the query objects. Using Arndt and Chang notation, edits to these portions of the string are evaluated to recreate the string, and $r_1 \cdots q_{f-1}$ must be parsed to obtain the ranks of query elements. The algorithm proposed here does not require parsing of the 2D string as changes to spatial relationships of the query objects are read directly from the edit sequence notation. This allows edits which do not affect the query objects to be discarded.

Our notation is similar to the Arndt and Chang notation in that the sequence is described by a string representing the first frame in the video, and a sequence of edits to this initial string. The edit operations, however, differ and are outlined below:

u	the next sequence of edits should be performed on the u axis of the string
v	the next sequence of edits should be performed on the v axis of the string
$m\ i, j$	move a symbol from position i to position j
$d\ i$	delete the symbol in position i of the u string, also deletes the corresponding end point in the u string and both symbols in the v string
$i\ ch, i$	insert the character ch into the string at position i
*	the previous edit completed the transformation to the representation of the next frame in the video

The sequence in figure 2 can be expressed as the following sequence of B–strings:

table box BOX TABLE, table TABLE box BOX
man MAN table box BOX TABLE, man table TABLE box BOX MAN
table TABLE man box MAN BOX, man table box BOX TABLE MAN

which is represented in our notation as:

table box BOX TABLE,table TABLE box BOX
u *i man*, 0 *i MAN*, 1 **v** *i man*, 0 *i MAN*, 5*
u *m* 2, 0 *m* 5, 1 *m* 3, 4 **v** *m* 2, 4*

Comparison of this edit sequences and the sequence in figure 2 show that the B–string based representation requires more space than the Arndt–Chang representation. This is to be expected as B–strings encode extent

information. In their paper Arndt and Chang give a space complexity analysis for a typical 2D string sequence. In their analysis the formula $\mid \log_2(\mid V \mid + \mid A \mid + n_1 + n_2 + 1) \mid$ is used to calculate the number of bits required to represent each token. A token is either an object label, operator, frame number or set number. In this formula V represents the set of object labels, A the set of operators, n_1 the number of frames in the sequence and n_2 the number of distinct sets. Arndt and Chang derive the following size for their representation of the sequence:

$$\begin{aligned} \text{Tokens needed for compressed set representation} &= 27 \\ \text{Bits required} &= 27 * \mid \log_2(\mid V \mid + n_1 + n_2 + 1) \mid \\ &= 243 \end{aligned}$$

The analysis for the same sequence using the B–string based representation, where T in the number of tokens and n_3 is the maximum length of the string is:

$$\begin{aligned} \text{Compressed B–string bits required} &= T * \mid \log_2(\mid V \mid + \mid A \mid + n_3 + 1) \mid \\ &= 53 * \mid \log_2(8 + 7 + 6 + 1) \\ &= 265 \end{aligned}$$

This result is favourable given the additional properties of the B–string representation. Table 1 gives the results from encoding the database of videos used for this project. The table gives the number of frames and objects in each sequence , along with space required to store the video sequence, the space taken by both a simple sequence of B–strings and the compressed B–string notation. The final column gives the size reduction achieved by using the compressed notation. The video sequences were stored and indexed on a Silicon Graphics workstation, using SGI–MVC 1 compression.

Sequence	Frames	Objects	Size (Kb)	B–strings	Edit sequence	Reduction
Fishi	34	15	772330	2063	1464	29%
Forrplex	20	21	1060880	1247	1006	19%
Pinex	31	22	2012602	2900	1704	41%
Swing	50	20	3294516	4348	2562	41%

Table 1: Compression results

In general we can estimate the size of the edit sequence required to encode a video sequence, using observed frequency of operators to predict the composition of the edit sequence. The average values for the occurrence of each operator observed in the sequences in the example database are shown in table 2. The seven operators may be represented using three bits. The number of bits required to encode a position in the string can be estimated as $l = \log_2(2.4 \mid O \mid) + 1$, where O is again the set of objects in the reference string. We use the figure 2.4 to estimate the number of symbols in the string. The number of symbols will be at least twice the number of objects, one symbol for each start and end point for an object, plus a number of equality symbols. The example sequences all display a length of less than 2.4 times the number of symbols per string. The number of bits required to represent each edit is given in table 3. Using the observed operator composition this yields a weighted average of

Edit type	Frequency
Move	42%
Axis	16%
Frame mark	8%
Insert	20%
Delete	14%

Table 2: Operator frequency

Edit type	Bits
Move	$3 + l + l$
Axis	3
Frame mark	3
Insert	$3 + \log_2 \mid O \mid + l$
Delete	$3 + l$

Table 3: Operator storage

$$\begin{aligned}\text{Number of bits} &= (0.42 * (3 + 2l)) + (0.16 * 3) + (0.08 * 3) + (0.2 * (3 + \log_2 \mid O \mid + l)) + (0.14 * (3 + l)) \\ &= 1.38 \log_2 \mid O \mid + 3.66\end{aligned}$$

Therefore the expected size of the edit sequence representation for a video of f frames, with n edits per frame would be

$$S = n \cdot f \cdot (1.38 \log_2 \mid O \mid + 3.66)$$

A sequence of 400 frames, with 20 objects and an expected 2 edits per frame would require approximately $S = 400 * 2(1.38 \log_2 \mid 20 \mid + 3.66)$ or 8448 bits. To store a video of this size using SGI MVC 1 encoding would require approximately 21 Mb. The example sequences in this paper contain an unusually high number of edits per frame, as the sample frequency is low in order to conserve storage on departmental machines.

4 QUERY RESOLUTION ALGORITHM

To make optimal use of storage space we would prefer to store data in a compressed format. In order to minimise response time we would prefer not to uncompress data as a preprocessing step. Ideally we would like to read the information required directly from its compressed form, and this facility is provided by the notation we propose.

The query resolution algorithm takes as inputs the edit notation for a sequence and a query string. The edit notation may be expressed as

initial u string
initial v string
$e_1 e_2 \ldots e_n$

where $e_1 e_2 \ldots e_n$ are the edits to the initial string. The query is a 2D B–string derived from the user's query. The query string is invariant throughout the matching process, so we can eliminate recalculation of the relationships between query objects by producing *relationship tables*. There are three types of relationship tables, one for each type of match.

For type–2 matching a relationship table is created containing the relationship category for each pair of objects. As there are five distinct relationship categories (see section 2), this table requires three bits for each element of the matrix, which is upper triangular as shown in table 4.

	o_1	o_2	o_3	$\ldots$	o_n
o_1		$r_{1,2}$	$r_{1,3}$	$\ldots$	$r_{1,n}$
o_2			$r_{2,3}$	$\ldots$	$r_{1,n}$
$\vdots$				$\ddots$	$\vdots$
o_{n-1}					$r_{n-1,n}$

Table 4: Relationship table

Type–1 matching requires relationship categories and also orthogonal relationship information, so a second relationship table is created to contain orthogonal relationships. As there are four orthogonal directions: north, south, east and west, we require four bits per entry. We also require orthogonal relationships for the reverse order of objects, as both *a orth_rel b* and *b orth_rel a* must be compared. We therefore keep two tables.

When a query requires type–0 matching a relationship table is created to store the exact relationship, along each axis, between each pair of objects. The 13 basic relationships require four bits for each axis, therefore

requiring one byte for each object pair. For n objects the space consumed by relationship tables, for each type of matching is:

$$\text{Type–2} \quad 3 \cdot \frac{n^2}{2} \text{ bits} \tag{1}$$

$$\text{Type–1} \quad (3+8) \cdot \frac{n^2}{2} \text{ bits} \tag{2}$$

$$\text{Type–0} \quad (3+8+8) \cdot \frac{n^2}{2} \text{ bits} \tag{3}$$

For 40 objects this represents a space requirement of less than 2kb.

The initial string from the edit notation is used to instantiate the *object rank table.* This table contains an entry for each query object and records the position within the reference string of the begin and end symbols along each axis, and the rank of each symbol. Once the object rank table has been instantiated we may proceed to match the first frame from the reference sequence.

Matching uses a *match table* (figure 3), which has a one bit entry for each object pair in the query. This entry is set if the object pair are a type–n match in the query and reference string. The match is determined by comparing relationships in the appropriate relationship table, or tables, with the relationship determined from the object rank table for the object pair. Thus for type–2 matching for query object o_i and o_j, we would compare the type–2 relationship table entry $r_{i,j}$ with the relationship category determined using the ranks from entries o_i and o_j in the object rank table. For all objects o_i and o_j that are found to match, we set entry $r_{i,j}$ in the match table, otherwise entry $r_{i,j}$ is cleared. A matching frame will result in all bits in the match table being set.

Once initial matching is completed we proceed to the next frame. To match the next frame we take the sequence of edits $e_1 e_2 \ldots e_m$, where e_m is the first frame completion marker, and from these edits modify the entries in the object rank table. Using the stored positions within the reference string of query object symbols we can clearly discard some edits immediately. If we use q_{min} to denote the first occurrence of a query symbol along an axis, and q_{max} the last, then the following edits do not affect query object relationships:

$$\begin{array}{ll} \text{m } i,j & \text{if } (i < q_{min} \vee i > q_{max}) \wedge (j < q_{min} \vee j > q_{max}) \\ \text{d } i & \text{if } i < q_{min} \vee i > q_{max} \\ \text{i } ch,i & \text{if } i < q_{min} \vee i > q_{max} \end{array}$$

In some of these cases the true ranks of query objects will be altered, but their relative ranks will remain the same. It is the relative positions that are required for this algorithm. In each of the above cases, we simply modify the current reference string to reflect the edit, then continue to the next edit. It is necessary to keep the reference string for cases such as moving a symbol from outside the range of the query, to inside the query range. Here we need to know whether the symbol is an object symbol, or an equality symbol. The reference string however is at no stage processed to determine either object ranks or symbol positions. Modifications to object ranks and symbol positions are read directly from the edit sequence.

When a frame completion marker is reached the matching table is updated. A frame match is detected if all bits in the match table are set. The process of updating the object rank table and match table is then repeated for the remaining portions of the edit string, with each matching frame being added to a match list.

5 OPTIMISATIONS TO QUERY RESOLUTION

There are two optimisations we can make to the matching process that can greatly improve performance. We first note that generally, not all objects are in motion. This implies that it is unlikely that all object relationships will change between frames. If not all relationships change then we need not update the entire match table after

each group of edits. Let $Q = q_1 \ldots q_i \ldots q_j \ldots q_n$ be the query, where $q_i \ldots q_j$ are the objects affected by a sequence of edits. Any match table element that does not involve an object in $q_i \ldots q_j$ will not have changed, and therefore need not be updated. The match table is displayed in figure 3, with elements that require updating shaded.

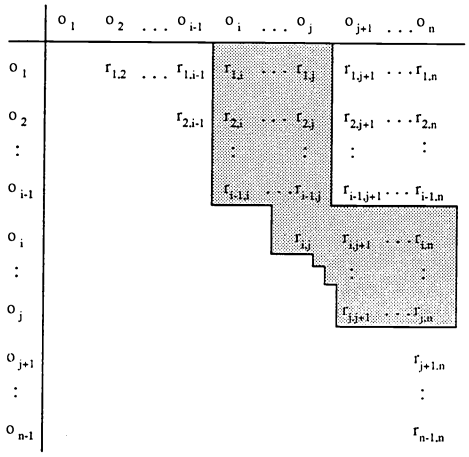

Figure 3: Match table

The second optimisation allows us to avoid updating the match table entirely in some cases. To implement this we must keep an extra bit for each row of the match table. This bit is used to indicate whether the row contains a mismatch. For the matrix in figure 3, if a row containing a mismatch occurs after row j it will not have been altered by the previous edits. This of course means the row will still contain a mismatch, and therefore there is no need to update the match table as the frame cannot match. We therefore continue and process the subsequent edits. This optimisation may not be used when performing similarity retrieval as similarity retrieval requires the match table for production of an association graph for maximal common subpicture detection.

If query objects are specified by attribute values, rather than unique object labels, there may be a number of query strings to attempt to match for a single query. In this case we can reduce computation by creating the relationship tables for each query string before beginning the matching process. We also create relationship tables for the reference string of each frame as come to match it. Matching may then be performed by comparing the reference frame relationship tables against each of the query tables. This allows matching with a single parse of the edit sequence, and without recalculation of relationships in the reference string. It is unnecessary to produce relationship tables for reference frames when only comparing against a single query table, as the cost of comparison is equivalent to the cost of creating the relationship tables for the reference string.

6 CONCLUSION

2D strings have been successful for indexing pictorial databases. In later work Arndt and Chang proposed an application of 2D strings to compression of indices for video sequences. This work provides a means of compression for a sequence of 2D strings, where each 2D string describes a single frame of a sequence. There are algorithms provided for conversion from a sequence of strings to compressed notationand the reverse transformation. Since this work was completed, 2D strings for spatial reasoning have increased considerably in sophistication.

The work presented here uses 2D B–strings, which provide a much improved representation of spatial information. Further, we considerably extend the earlier work by providing a method for reading relationship changes directly from the compressed notation. This alleviates the need for decompression of the notation as a preprocess-

ing step. There is also a matching algorithm which takes advantage of the properties of the new representation to reduce computation. Thus this system provides a compact and efficient indexing scheme for video data.

Further work in this area has produced a three level scheme for video sequence to video sequence matching. This uses the work presented here to define frame matches and provides definitions of matching between a sequence of iconic frames and a video sequence. This work is now being applied to various problems in multimedia.

7 ACKNOWLEDGEMENTS

This work is supported in part by an ARC grant number 4392.

8 REFERENCES

[1] A. Akutsu, Y. Tonomura, H. Hashimoto, and Y. Ohba. Video indexing using motion vectors. In *SPIE Proceedings of Visual Communications and Image Processing '92*, volume 1818, pages 1522–1530. SPIE, 1992.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[3] T. Arndt and S.-K. Chang. Image sequence compression by iconic indexing. In *1989 IEEE Workshop on Visual Languages*, pages 177–182. IEEE, IEEE Computer Society, October 1989.

[4] A. S. Chakravarthy. Towards semantic retrieval of pictures and video. In *Indexing and Reuse in Multimedia Systems*, pages 12–18, Seattle, WA, August 1994. AAAI, AAAI. Workshop Notes.

[5] S. Chang, E. Jungert, and T. Li. Representation and retrieval of symbolic pictures using generalized 2D strings. In *SPIE Proceedings of Visual Communications and Image Processing IV*, volume 1199, pages 1360–1372. SPIE, 1989.

[6] S. Chang, Q. Shi, and C. Yan. Iconic indexing by 2D strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[7] S. K. Chang, C. W. Yan, D. C. Dimitroff, and T. Arndt. An intelligent image database system. *IEEE Transactions on Software Engineering*, 14(5):681–688, May 1988.

[8] M. Davis. Knowledge representation for video. In *Indexing and Reuse in Multimedia Systems*, pages 19–28, Seattle, WA, August 1994. AAAI, AAAI. Workshop Notes.

[9] E. Jungert. Extended symbolic projections as a knowledge structure for spatial reasoning. In *4th BPRA Conference on Pattern Recognition*, pages 343–351. Springer Verlag, March 1988.

[10] S. Lee and F. Hsu. Spatial reasoning and similarity retrieval of images using 2D C-string knowledge representation. *Pattern Recognition*, 25(3):305–318, 1992.

[11] S. Lee, M. Yang, and J. Chen. Signature file as a spatial filter for iconic image database. *Journal of Visual Languages and Computing*, 3:373–397, 1992.

[12] K. Otsuji, Y. Tonomura, and Y. Ohba. Video browsing using brightness data. In K. Tzou and T. Koga, editors, *SPIE Proceedings of Visual Communications and Image Processing '91: Image Processing*, pages 980–989, Boston, MA, November 1991. SPIE.

[13] H. Tamura and N. Yokoya. Image database systems: A survey. *Pattern Recognition*, 17(1):29–43, 1984.