IDIAP RESEARCH REPORT

IDIAP

Martigny - Valais - Suisse

VIDEO SEQUENCE MATCHING VIA DECISION TREE PATH FOLLOWING

Kim Shearer [a] Svetha Venkatesh [b]
Horst Bunke [c]

IDIAP–RR 00-12

À PARAÎTRE DANS
Pattern Recognition Letters

Institut Dalle Molle d'Intelligence Artificielle Perceptive • CP 592 • Martigny • Valais • Suisse

téléphone +41 – 27 – 721 77 11
télécopieur +41 – 27 – 721 77 12
adr.él. secretariat@idiap.ch
internet http://www.idiap.ch

a IDIAP
b School of Computing, Curtin University of Technology, Perth, Australia
c Institut für Informatik und Angewandte Mathematik, Universität Bern, Switzerland

Rapport de recherche de l'IDIAP 00-12

Video sequence matching via decision tree path following

Kim Shearer Svetha Venkatesh Horst Bunke

à paraître dans
Pattern Recognition Letters

Résumé. This paper presents an algorithm for resolution of a sequence of incrementally changing iconic queries, against a known database of model graphs. The algorithm is based on a representation using graphs and subgraph isomorphism detection.

1 Introduction

Image retrieval has been tackled with a significant degree of success by a number of groups (Flickner *et al.*, 1995; Lam *et al.*, 1995; Holt and Hartwick, 1994). While this technology is employed commercially, video retrieval is still undergoing development towards a similar position.

The main difficulty with video databases is finding an effective query paradigm. Whereas image databases can be treated as static data, video contains temporal change which must be accessible to a retrieval system. The usual image database approach is query by pictorial example, where a sample picture is presented and some set of attributes is used to determine which images from the database are most similar. A ranked list of similar retrieved images is then presented to the user. A number of novel approaches for automatic video database retrieval, such as XSTL by Del Bimbo *et al.* (1995) and Smith and Kanade (1998) have been proposed. However, these systems do not offer a significantly better retrieval scheme than the extended query by pictorial example approach.

Query by pictorial example, used extensively in pictorial databases, may be extended to video by providing a sequence of iconic queries, representing a temporal sequence of states of interest for retrieval. Each state in the sequence may be mapped to zero, one or more frames in a video sequence. Differing levels of exactitude for matching are defined, to allow approximate retrieval when exact matches are not present, such as in Shearer *et al.* (1997a). One representation commonly used in pictorial databases is indexing by qualitative spatial relationships, which often uses iconic example as the query mechanism. The query is constructed by positioning icons representing objects, or classes of object, within a picture. A video query consists of a sequence of such iconic examples which define the sequence of changes to the relationships between objects over time. Such a query is resolved by finding a set of models from the database to match each of the states in the sequence of changes. Conditions of length and contiguity can then be used to determine which models from the sets retrieved form a matching sequence.

If queries are expressed in this form it is necessary to determine sets of matching video frames for each step of the iconic query sequence. An elegant and natural way of expressing image similarity is as graph isomorphism detection.

Graphs provide a highly expressive representation for relational structures, for which there is a large body of well studied algorithms. However, while graph and subgraph isomorphsims are a useful expression of similarity between relational structures, the standard algorithms to detect isomorphism have high computational complexity. The algorithm commonly used to detect exact subgraph isomorphism between two graph is due to Ullman (1976), which for a pair of graphs G_1 and G_2 with numbers of vertices m and n, has complexity $O(m^n n^2)$. Messmer and Bunke (1997) have addressed the subgraph isomorphism problem from the view point of classifying input graphs, given on line, against a database of models graphs of which there is prior knowledge. This type of problem occurs in a number of classification areas beside image and video retrieval (e.g. Shearer *et al.*, 1998b; Lam *et al.*, 1995; Rouvray and Balaban, 1979).

One of the algorithms proposed in Bunke and Messmer (1997) is based on the construction of a decision tree from the database of model graphs. This algorithm has time complexity of $O(n^2)$, where n is the number of vertices in the input graph. Previous work has extended this algorithm to detection of the largest common subgraph between an input graph and a database of model graphs (Shearer *et al.*, 1998a). This paper presents an extension to the decision tree algorithm for the detection of graph and subgraph isomorphisms from a sequence of input graphs, to a known database of model graphs. The sequence of input graphs is created by incremental changes to the state represented by the graph.

A relational structure which changes incrementally over time can be represented by an initial state, plus edits to that state representing the incremental changes. If the initial state is represented using a graph, then the changes will be graph edit operations. The problem addressed by the algorithm presented in this paper is that of detecting a set $S_i = \{M_{i_1}, M_{i_2}, \ldots, M_{i_k}\}$ of model graphs for each graph G_i in the sequence of changes, where each M_{i_j} contains a subgraph isomorphic to G_i. This correspondence is depicted in Figure 1. The simplest approach to solving this problem is to classify

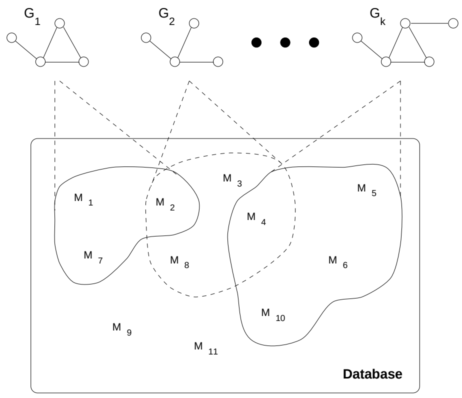


FIG. 1 – *The correspondence between a sequence of dynamic graphs and a model database.*

each graph in the changing sequence individually, however the new algorithm is considerably more efficient.

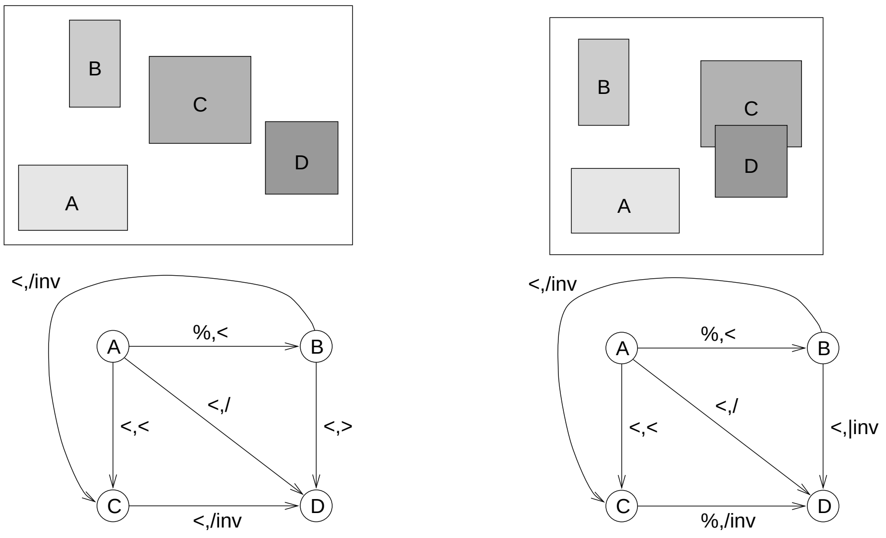
The next section of this paper gives a brief background on encoding of images and video as graphs, for similarity retrieval by isomorphism detection. The paper continues with a description of the decision tree based graph isomorphism algorithm of Bunke and Messmer (1997). The new algorithm to extend this to dynamically changing graphs is then presented. This new algorithm avoids complete classification of any state except the first by using the graph edit operations to navigate a path through the decision tree. The final sections contain experimental results which show the advantage of the new algorithm, and a summary of the conclusions of this work.

2 Encoding videos

The representation of video as a relational structure can take advantage of the work done with pictorial databases, such as the work of Chang *et al.* (1987, 1989) and Lee and Hsu (1990, 1992). Taking one of these as a base representation, video can then be encoded by incorporating temporal changes into the notation as suggested by Arndt and Chang (1989) and Shearer *et al.* (1997c). In this work the state for the initial frame of the video is represented in full. This involves representing the qualitative spatial relationships of the objects in the initial frame. The representation of the frame is referred to as a state because the qualitative spatial relationships between the objects indexed may not change for a number of frames, thus a single state may represent more than one frame.

A video sequence can then be represented by a full encoding of the initial frame, followed by a sequence of edits to the prevailing state to represent subsequent frames. The sequence of edits is grouped into sets, each set taking the state representing one frame, or set of frames, to the state representing the following frame or set of frames. Due to one state usually representing a number of frames the encoding is already compact. In addition, temporal subsampling or the suitable choice of key frames, rather than encoding all frames, can further compress the encoding. When this encoding is interpreted using graphs for the purpose of similarity retrieval, it maps naturally to an initial graph corresponding to the initial state, and a sequence of graph edit operations to this graph.

The usual encoding of spatial relationships in pictorial information as graphs is to use vertices of the graph to represent objects, and labelled edges to express the spatial relationships between objects. If the relationship symbols proposed by Allen (Allen, 1983) are employed, the two simple pictures in Figure 2 can be expressed as shown. To see the relationship between pictorial similarity and graph isomorphism, observe that the object D in Figure 2 changes its qualitative relationship to object C, while the other objects retain the same qualitative relationships. Examination of the two graphs reveals that if the vertex labelled D and its incident edges were to be removed, the remaining sections of the two graphs would indeed be isomorphic.



(a) (b)

FIG. 2 – *Two pictures with possible graph encodings*

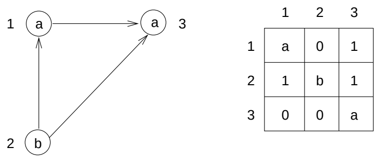


FIG. 3 – *Graph with adjacency matrix*

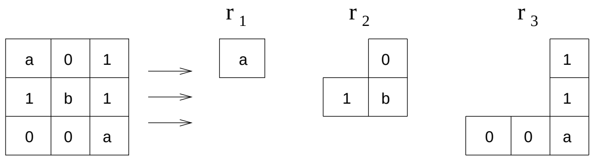


FIG. 4 – *Row column elements of a matrix*

In similarity retrieval of models from a database of pictorial information the models to be retrieved are those with the largest subgraph isomorphic to the query input. The next section of the paper outlines an algorithm for finding subgraph isomorphisms from the query input to the models in a database.

3 Decision tree algorithms

A subgraph isomorphism from a graph G_1 to a graph G_2 is defined as an injective mapping of vertices from G_1 onto a subset of the vertices of G_2, such that structure is preserved. The algorithm described in this section detects subgraph isomorphisms from the input graph to the model graphs. Given the definition we see that this implies the detection of subgraphs within models which are isomorphic to the input graph. Consideration of the requirements of iconic query by example reveal that this is an appropriate form of isomorphism detection for such a problem.

The decision tree is constructed using the adjacency matrix representation for graphs. Figure 3 shows a graph and an adjacency matrix which can represent it. In the matrix the diagonals contain the labels of the vertices, for the remainder of the matrix element e_{ij} contains the edge label for the edge between vertex i and vertex j. In this example the edges are unlabelled, and so the entry 1 represents the presence of an edge, while 0 represents the absence of an edge.

The decision tree algorithm stems from the property of adjacency matrices that each permutation of an adjacency matrix M_1, representing a graph G_1, gives a matrix representing a graph isomorphic to G_1. Thus for a permutation matrix P, the matrix M' where $M' = PMP^T$ represents a graph isomorphic to the graph represented by M. The detection of graph isomorphisms between two graphs G_1 and G_2, represented by adjacency matrices M_1 and M_2, is therefore equivalent to finding permutation matrices which transform the matrix M_1 to M_2.

This equivalence is useful as it is possible to create a decision tree from the adjacency matrix for a graph, and the permutations of that adjacency matrix, which allows classification of input graph with computational complexity independent of the size and number of model graphs in the database. This decision tree is constructed using the *row column elements* of the adjacency matrix. Each square matrix M with n rows can be broken into n row column elements $RCE = \{r_1, \ldots r_n\}$, where $r_x = \{M_{xj} : 1 \leq j \leq x \wedge M_{ix} : 1 \leq i \leq x\}$. Figure 4 shows the adjacency matrix presented earlier in Figure 3, separated into row column elements. A row column element r_i, therefore, contains the label of the i^{th} vertex, and the labels of all edges between the i^{th} vertex and all vertices which precede it in the matrix. A decision tree can be constructed with an unlabelled root node, with one descendant for each unique one element RCE in the set of permutation matrices for a graph. Thus in Figure 6 there is an adjacency matrix and the five different permutations, with two unique labels, a and b, giving two descendants from the root node. The edges are labelled with the RCE, with the root node termed level 0, and level number increasing with distance from the root. This gives level 1 of the decision tree with nodes $L1_1, \ldots L1_i$, each edge being labelled with a size 1 row column element from an adjacency matrix. Once level k has been completed for the decision tree, level $k+1$ can be added by taking, for an RCE r_j represented on level k, all RCE of size $k+1$ which follow r_j in at least one adjacency matrix and introducing a node for each. The edges to the new nodes are labelled with the size $k+1$ row column elements. Figure 6 shows the decision tree generated from the six possible adjacency matrices for the graph of Figure 3.

The isomorphism detection process follows a similar pattern to the construction process for the decision tree. An input graph G_I may be classified against a decision tree by taking the associated adjacency matrix M_I, and navigating the tree using the row column elements of this matrix. Thus, beginning at the root, the first one element RCE is compared to the labels on the initial edges of the tree, and classification proceeds to the node at the end of the matching edge if one exists. This process continues as long as there is a match for the next RCE of M_I from the current node in the tree. The two termination conditions are:

1. No edge label matches the next RCE, this is termination with failure.
2. All RCE in M_I have been used, indicating success.

If all RCE are used then the model graphs associated with the final node reached in the decision tree contain an subgraph isomorphism to the input graph G_I.

This classification process gives an algorithm which can detect subgraph isomorphisms from the input to the models graphs with computational complexity of $O(n^2)$, where n is the number of vertices in the input. This complexity is independent of the size of the model graphs and the number of model graphs in the database, which is clearly an advantage for query by iconic example where the input will generally be considerably smaller than most models.

4 Navigating for dynamic matching

The extension of the basic decision tree algorithm presented in this work is for the detection of isomorphisms between a sequence of input graphs and a database of model graphs. An example of the form of this work is illustrated in Figure 1. In this figure there is a sequence of input graphs G_1,G_2,$\ldots G_k$, and a database of model graphs $M_1 \ldots M_{11}$. Each of the input graphs in the sequence will be a subgraph isomorphism for a subset of the database, this imposes set of models, S_1 being the set of all models with a subgraph isomorphism to input graph G_1, and likewise for $G_2 \ldots G_k$.

The sequence of input graphs is encoded as a full graph representing the initial state, and a string of graph edit operations. Contained within the string are typical graph edit operations, such

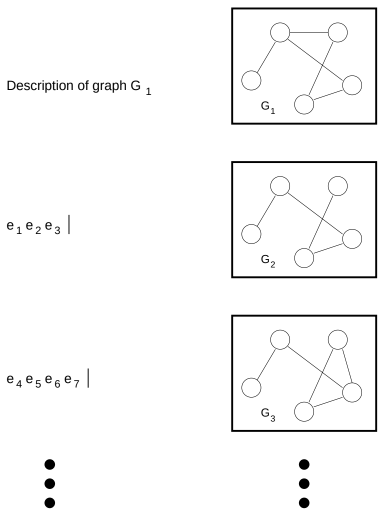


FIG. 5 – *Format of input graph sequence*

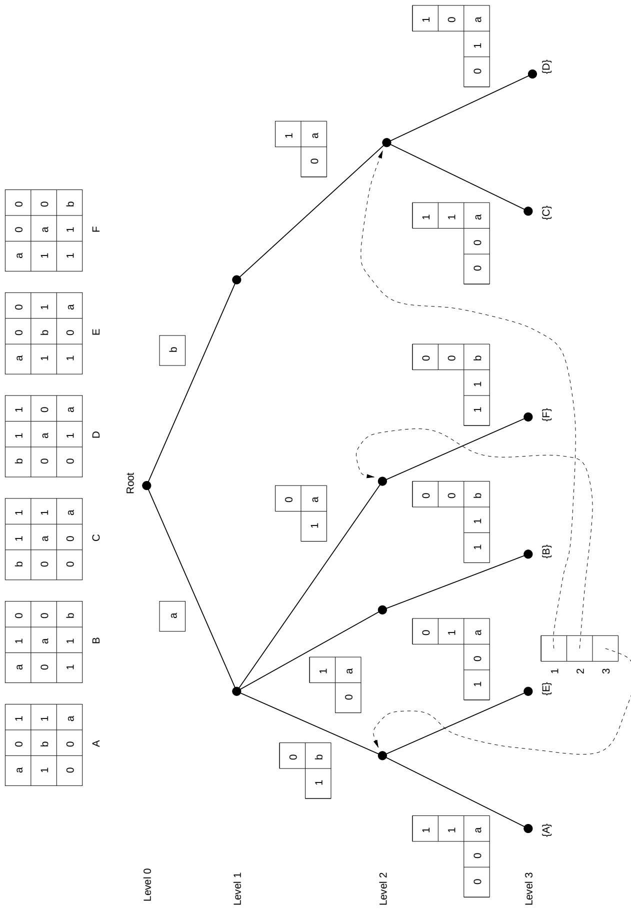


FIG. 6 – *Decision tree with pointers for RCE deletion*

as insertion, deletion and change of label for vertices and edges, and also frame separators. A frame separator appears in the edit operator string when the sequence of edits to the current point has completed the transformation of one graph G_i to the subsequence graph G_{i+1}. Figure 5 illustrates this, with edits e_1, e_2 and e_3 required to transform G_1 to G_2, and edits e_4, e_5, e_6 and e_7 transform graph G_2 into graph G_3.

The algorithm begins with a classification of the initial state graph, using the algorithm described in the previous section. The node reached in this initial classification, N_1, represents the models which contain an isomorphism to G_1. The models associated with N_1, therefore, become the set S_1 of models containing subgraphs isomorphic to G_1.

To continue the algorithm, assume we have a node N_i and a set S_i for the graph G_i. The algorithm continues using the substring of edits which transform G_i to G_{i+1}. Initially consider insertion and deletion of vertices as edit operations. These operations may be accommodated using one of the cases:

Case 1 Insertion of a new vertex. The RCE for the new vertex is added to the adjacency matrix as the final RCE, and classification continues by searching for a descendent of decision tree node N_i with an arc label matching the new RCE. The possible outcomes of this are:

1. A descendent arc from N_i is found which is labelled with the new RCE, and leads to a node N_j. If the next edit operation in the string is a frame completion marker, N_j becomes N_{i+1} and models associated with N_j become the set S_{i+1}. Classification will continue from N_j.
2. No descendant arc from N_i is labelled with the new RCE. This implies there is no subgraph isomorphism for the new graph. If the next edit operation is a frame completion marker

$$\begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & a & 0 & 0 \\ 2 & 1 & b & 1 \\ 3 & 1 & 0 & a \\ \hline \end{array}$$

E

(a) Adjacency matrix

$$\begin{array}{c|cc|} & 1 & 3 \\ \hline 1 & a & 0 \\ 3 & 1 & a \\ \hline \end{array}$$

(b) Adjacency matrix, second RCE deleted

FIG. 7 – *Example of RCE deletion for the decision tree pointer redirection*

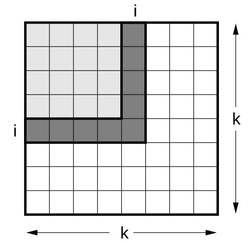


FIG. 8 – *Adjacency matrix separation for creation of pointers*

then $S_{i+1} = \emptyset$. Classification continues from N_j.

Case 2 Deletion of an existing vertex. Assume the vertex to be deleted is the last vertex in the adjacency matrix, then the edit simply causes a step back up the tree to the immediate ancestor on the node N_i. If the next operator is a frame completion marker, then the model graphs associated with the ancestor node become the set S_{i+1}.

By this process, a sequence of edits operations which are either an insertion, or a deletion of the vertex of the final RCE may be completed. If the edit string contains l frame completion markers a sequence of model sets $\{S_1 \ldots S_l\}$ will result. In the application for which this algorithm has been developed, each set is a collection of frames from a video database.

While not all deletions will occur from the final RCE, it is possible to generalise deletion by the addition of a small amount of structure to the decision tree. Each node at level k of the decision tree has k possible subgraphs with $k-1$ vertices, each of which is represented in the decision tree. In order to accommodate the deletion of RCE i, where $1 \leq i \leq k$, there are k pointers added to each node at level k, which point to the correct level $k-1$ node of the decision tree for each deletion. An example of these additional pointers is given for one node in Figure 6. The adjacency matrix for the node representing $\{E\}$ has three RCEs, so there are three pointers, the simplest being a pointer to the immediate ancestor for deletion of the final RCE. These pointers may be included in the tree off–line at creation time, and so do not influence classification time.

4.1 Pointer Creation

The algorithm for decision tree construction employs a breadth first order for node creation. This ordering is assumed for the addition of the pointers for RCE deletion, as it ensures that all subgraphs at level $k-1$ are present before the addition of any nodes at level k. Location of the correct node for each RCE deletion from a node N_c is undertaken as follows.

From node N_c, with adjacency matrix A_c having k rows and columns, it is necessary to find the correct node N_r on level $k-1$ for classification to continue if the i^{th} RCE is detected. This node N_r represents the adjacency matrix with $k-1$ rows and columns that is formed if RCE i is deleted from A_c. To illustrate this, Figure 7(a) gives the matrix labelled $\{E\}$ from Figure 6, and Figure 7(b) gives the adjacency matrix which is the result of deleting the second RCE. Reference to Figure 6 shows that the second redirection pointer does indeed point to the node representing the matrix of Figure 7(b). Figure 10 shows the graphs represented by the two adjacency matrices from figure 7, where the second RCE has been deleted, representing the deletion of the vertex labelled b.

In order to locate the node N_c most efficiently consider the adjacency matrix of Figure 8. The shaded section of the matrix above the deleted RCE i does not change when the deletion is performed. Therefore, to find N_c we need step back only $k-i$ ancestors, retaining the classification to this point. Classification may then continue at RCE $i+1$, and descend the remainder of the adjacency matrix. This descent will terminate at the node in level $k-1$ which represents the adjacency matrix A_c with the RCE i deleted.

a	0	0	1
1	b	1	0
1	0	c	0
0	1	0	b

(a) Original adjacency matrix

a	0	0	1
1	b	1	0
0	0	c	0
0	1	0	b

(b) Modified element in the adjacency matrix

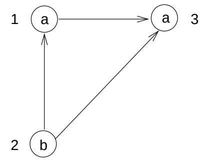
a	0	1
1	b	0
0	1	b

(c) Adjacency matrix with RCE deleted

a	0	1	0
1	b	0	1
0	1	b	0
0	0	0	c

(d) Permuted adjacency matrix

FIG. 9 – *Adjacency matrix manipulation during the edge deletion process*



(a) Graph from adjacency matrix in figure 7(a)

(b) Graph from adjacency matrix in figure 7(b)

FIG. 10 – *Graphs represented by adjacency matrices in figure 7*

The addition of these pointers to facilitate RCE deletion makes vertex insertion and deletion possible in one step. This leaves edge insertion and deletion, and label substitution to be dealt with. These remaining operations can be considered equivalent in that each may be performed as a single vertex deletion, followed by a vertex insertion. To perform an edge change or a label substitution, the RCE containing the matrix element e_c which must change is deleted, then the RCE with e_c altered, is inserted. This is practical due to the low cost of RCE deletion.

The algorithm for label substitution and insertion and deletion of edges proceeds as follows.

1. Follow the pointer for deletion of the i^{th} RCE to the appropriate node on the previous level of the decision tree.
2. Perform the modification in the i^{th} RCE. For a label substitution the label is changed, for an edge addition the label of the new edge is inserted into the appropriate matrix element, and for an edge deletion a null label is placed in the matrix element representing the deleted edge.
3. Permute the i^{th} RCE to the bottom of the input adjacency matrix.
4. Continue from the redirected node found in step 1 to classify the modified RCE in the last RCE of the adjacency matrix.

Figure 9 shows a simple example of this algorithm. The Figure 9(a) shows the original adjacency matrix and Figure 9(b) shows the change required. The adjacency matrix with the third RCE element deleted is given in Figure 9(c), and the matrix with the new RCE appended is shown in Figure 9(d).

Thus, using this algorithm, at most one redirection and one RCE classification are required to complete any label substitution or edge insertion or deletion.

5 Results: efficiency of retrieval

The times quoted in this results section have all been obtained on an IBM compatible PC, with a Pentium II–266 processor and 96 Mb of main memory. The number of models in the database has been

	Number of Models	Model range	Input size	DG range	Number of Operations	Dynamic tree execution time	Initial graph classification time
1	38	9–10	9	7–9	40	0.18	0.16
2	57	8–10	9	7–9	40	0.31	0.28
3	57	8–10	9	7–9	200	0.38	0.28
4	93	4–10	9	7–9	200	0.26	0.16
5	214	4–10	9	7–9	200	0.58	0.50
6	57	8–10	6	5–7	40	0.09	0.08
7	57	8–10	6	5–7	100	0.11	0.08
8	57	8–10	6	5–7	150	0.12	0.08
9	57	8–10	6	5–7	200	0.14	0.08
10	57	8–10	6	5–7	300	0.18	0.08
11	214	4–10	6	5–7	100	0.85	0.80
12	214	4–10	6	5–7	150	0.87	0.80
13	214	4–10	6	5–7	200	0.89	0.80
14	214	4–10	6	5–7	300	0.93	0.80

TAB. 1 – *Execution time for the dynamic tree algorithm and a single classification*

limited so that the entire database was able to fit in memory, eliminating any memory management influences.

The results of the experiments are split into two groups based on number of vertices in the query. For each size of query a range of numbers of models in the database, and a range of numbers of graph edit operations are presented. The statistics presented for each experiment in Table 1 are:

1. The number of models contained in the database. That is the number of model graphs compiled into the decision tree.
2. The range of the number of vertices for the model graphs in the database. This is presented as two hyphen separated numbers, *min–max*.
3. Number of vertices in the initial input query graph.
4. The range of number of vertices for the dynamic graph.
5. Number of edits operations performed during the dynamic graph test.
6. Time in seconds for the classification of the dynamic graph sequence.
7. Time in seconds for one complete classification of the initial input graph.

To gain an estimate of the time required to classify each dynamic sequence it is reasonable to multiply the number of edit operations by the final column, the time for classification of the initial graph. A brief examination of the time taken to classify the dynamic sequence using the new algorithm shows that classification of a sizable edit sequence takes little time compared to the cost of the initial classification.

The set of experiments $1 \ldots 5$ shows and increasing number of models and edit operations. As expected the time required for matching increases with increasing database size and number of operations. The decrease for the fourth experiment is due to the decreasing size of the models, from a minimum of 8 vertices to a minimum of 4 vertices. The first experiment reported in Table 1 shows an increase of only 12.5% for dynamic matching of 40 edit operations over the initial classification time. This is a considerable improvement over repeating classification 40 times, once for each graph in the sequence.

In fact all experiments performed show an increase of only a small fraction of the cost of a single match even for a sequence of 200 operations. Each of the remaining experiments involves a variation of one of the parameters for the classification process. The first five experiments involve an initial input of size 9, with randomly chosen edits, evenly mixed between insertions, deletions and substitutions. The parameters varied are the increasing size of the database and the increasing number of edit operations. The increases in the execution time of the dynamic graph algorithm are even and well behaved.

The final two sets of experiments ($6 \ldots 10$ and $11 \ldots 14$) involve a smaller initial graph, of 6 vertices, with the model database kept constant and four of five different samples of edit operations. Two

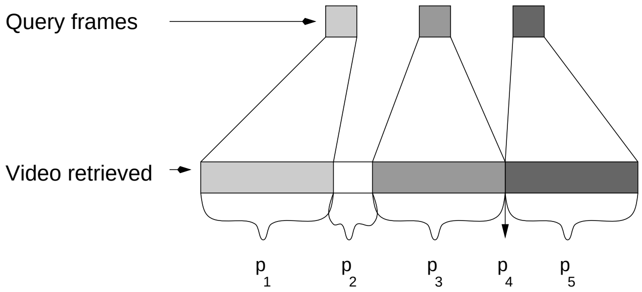


FIG. 11 – *Correspondence of query frames and video*

database sizes are presented for these experiments, with experiments 11 . . . 14 using a larger database than experiments 6 . . . 10. In all cases the dynamic graph algorithm performs efficiently and greatly reduces the time required to classify a dynamically changing graph sequence against database of model graphs. The time required for the dynamic portion of the classification increases in a well behaved fashion.

5.1 Results: illustration of retrieval

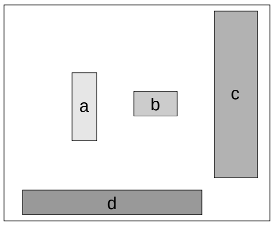
The application for which this algorithm has been developed involves matching a sequence of pictorial query by example frames against a database of video sequences. The user constructs each frame of the query by selecting a number of object labels, or object class labels from the list of labels available within the database. The selected labels are placed within the example frame as scalable icons, which the user may arrange to their satisfaction. The query by example contains a number of such frames, each showing the qualitative spatial relationships of key objects. The frames of the query are not usually matched to video sequences in a one to one contiguous fashion. Rather each query frame is taken to represent a state, which may correspond to one or more frames in the retrieved sequence. Permitting the user to specify the relative lengths of matching sections, and contiguity properties, provides a flexible query system. Figure 11 is a diagram of the matching process, indicating some of the parameters which may be varied.

During the matching process each frame in the query is matched against the states in the video database, each of which represents a set of frames in one or more videos. When a matching state is detected, the frames it represents contain the correct spatial relationships. The sets of matches detected for each query frame are then examined to see which can satisfy the contiguity conditions to form a suitable match for the query. Contiguity may be specified in a number of manners using parameters for the length of matching sections and non–matching sections. The lengths of matching sections are represented in Figure 11 by parameters p_1, p_3 and p_5. Non–matching sections have lengths p_2 and p_4. Possible contiguity conditions may be based the following parameter specifications:

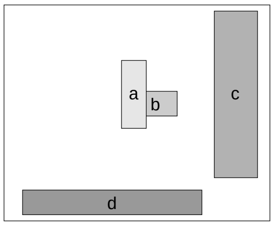
1. The last frame of one matching section must be contiguous with the first frame of the next matching section (e.g. p_4).
2. There may be a fixed number of non–matching frames allowed between two matching sections (e.g. $p_2 < T$).
3. The length of the non–matching section between two matching sections must be less than a threshold fraction of the lengths of the surrounding matching sections (e.g. $p_2 < R \cdot (p_1 + p_3)$).

Combinations of these matching schemes, such as a fixed maximum length of non–matching sections, with the local upper limit set using a fraction of sequence length, would also be reasonable. For a non–matching section p_i, with threshold T, the limit might be defined as follows:

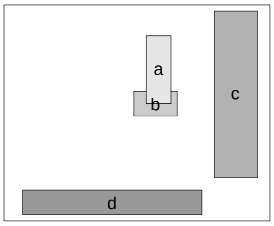
$$l_{max} = \begin{cases} R \cdot (p_{i-1} + p_{i+1}) & \text{if } R \cdot (p_{i-1} + p_{i+1}) < T \\ T & \text{Otherwise} \end{cases} \tag{1}$$



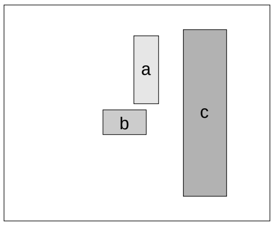
(a) Query frame 1



(b) Query frame 2

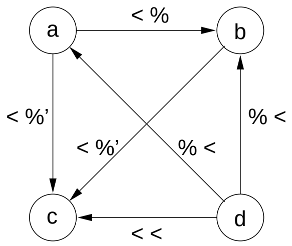


(c) Query frame 3



(d) Query frame 4

FIG. 12 – *Representation of the iconic query*



(a) Graph representing the initial frame

Label on $a \rightarrow b$ becomes [%

(b) Edits for frame 2

$a \rightarrow b$ with label [% becomes $b \rightarrow a$ with label % /

(c) Edits for frame 3

Label on $b \rightarrow a$ becomes / <.
Vertex d and incident edges are deleted

(d) Edits for frame 4

FIG. 13 – *Initial graph and description of the graph edits*

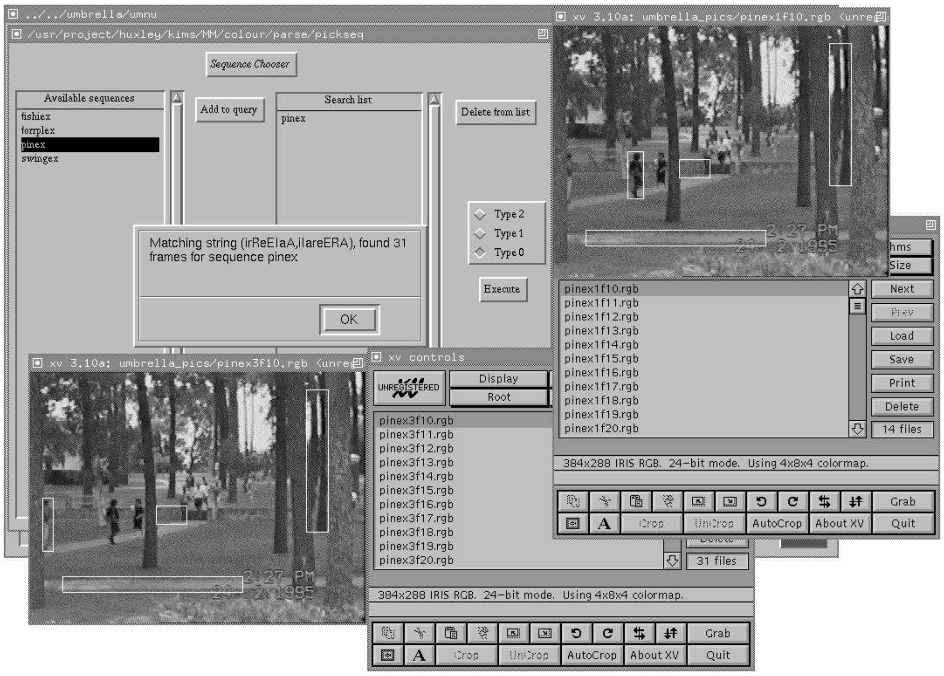
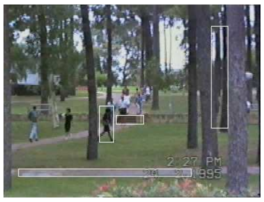
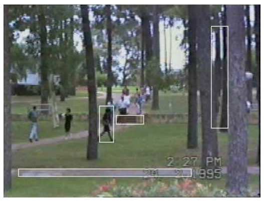


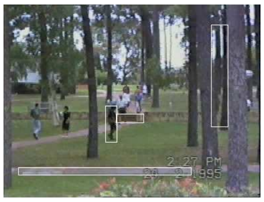
FIG. 14 – *Retrieval results from example dynamic input sequence*

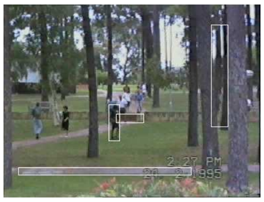

(a)

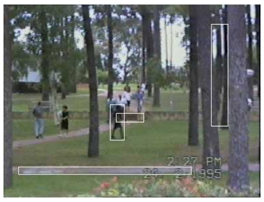

(b)


(c)


(d)


(e)


(f)


(g)

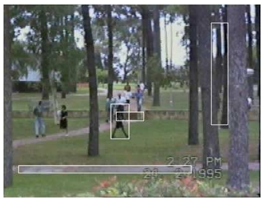

(h)

FIG. 15 – *Frames forming the retrieved match*

.

An example of retrieval using contiguity of matching sections is show in Figures 12, 13 and 14. Figure 12 shows the form of the iconic sketch used for querying the system, and Figure 13 shows the graph representing the initial state and describes the edits required. Figure 14 shows a screen image of the results presented by the retrieval system. There may be more than one set of objects which can form a matching sequence for the query in a single video sequence. For the video sequence retrieved in Figure 14 there are in fact two sets of objects which exhibit the correct spatial relationships to form a matching sequence. The match to the left of Figure 14 is an inexact match, with the object which is mapped to object a in the query exhibiting orthogonal (type–1) similarity, rather than exact relationship (type–2) similarity. An explanation of types of matching for video sequences can be found in Shearer *et al.* (1997a).

For each of the sets of objects which form a match an image viewer is presented, which lists the frames which form the sequence matching the query. Figure 14 shows the two image viewers, one for each of the sets of matching objects. Each viewer displays the first frame of the matching sequence, with the list of frames forming the sequence displayed alongside. A selection of eight frames from the sequence on the right in Figure 14 is shown in Figure 15. In this figure the first four frames match the initial frame of the query, while the remainder match the second query frame. Further details on the notation and retrieval system can be found in Shearer *et al.* (1997c). These frames are chosen to show the transition from one state to the next. In fact the retrieval sequence contains 16 frames matching the first query frame.

Applications of this work occur when the relative movement of objects is important in retrieval. One application uses this work to present video of a guide moving across a university campus Shearer *et al.* (1997a,b). The overall video is constructed from clips which represent sections of the path across campus, thus allowing a flexible interface. Each of the clips has certain key objects indexed and this allows reasoning about the most suitable clips for each section of the path, based on direction of travel and entry and exit of buildings. By retrieving clips based upon direction of motion and relative motion of the guide and key objects, the final video may be contructed from the sequence of automatically retrieved clips.

Another promising application is in traffic monitoring, where cameras mounted at intersections and along roads track vehicle motion. In this domain automated object segmentation is possible due to the highly constrained shape and motion of the key objects. Cars and other vehicles are rigid bodies and present largely plannar motion. Significant events in traffic flow, such as accdients and illegal manouvers are defined by the relative motion of the vehicles tracked, thus retrieval using spatial relationships is highly suitable. An preliminary investigation of this application using spatial reasoning can be found in Del Bimbo *et al.* (1995). Similar applications in security monitoring are more difficult due to the unstructured shape of objects, and the difficulty this presents for object segmentation.

Biological research, in the area of cell biology, also has a use for such an indexing system, in automated event recognition and query by content. In this work the movement and position of bacteria and cells are used as keys for retrieval and recognition of important events Shotton *et al.* (2000). Previous work has centered upon retrieval by characteristics of individual cells, such as cell velocity, and the tumble speed and direction of cells and cell groups. Current work seeks to employ the automatically extracted cell position and identification information, to identify sequences of cell interaction which signal an event of interest. The system described in this paper is one possibility being assessed for the rapid detection of key events in such a retrieval system.

6 Conclusions

Given the time complexity per edit operation of $O(n)$, compared to a reclassification time complexity of $O(n^2)$, we can state that if the number of edits required to transform one graph to the next in the sequence is in general less than n, the new algorithm should provide improved performance. In practice it can be seen that the performance of the dynamic isomorphism detection is far better than repeated classification of full graphs.

The results reported show that in practice the classification of edit operations using this algorithm is significantly less expensive than repeated reclassification of the sequence of input graphs. The addition of minimal additional structure to the decision tree has led to a highly efficient algorithm for detecting isomorphisms to incrementally changing input graphs. The application of this algorithm to video retrieval based on qualitative spatial relationships gives an empirical example of the advantages proposed.

Références

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

Arndt, T. and Chang, S.-K. (1989). Image sequence compression by iconic indexing. In *1989 IEEE Workshop on Visual Languages*, pages 177–182. The Institute of Electrical and Electronic Engineers, IEEE Computer Society.

Bunke, H. and Messmer, B. (1997). Recent advances in graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, **11**(1), 169–203.

Chang, S., Shi, Q., and Yan, C. (1987). Iconic indexing by 2D strings. in *Proceedings of the IEEE Workshop on Visual Languages,* Dallas, Texas, USA, June 1986. Also in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **9**(3), 413–428.

Chang, S., Jungert, E., and Li, T. (1989). 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.

Del Bimbo, A., Vicario, E., and Zingoni, D. (1995). Symbolic description and visual querying of image sequences using spatio temporal logic. *IEEE Transactions on Knowledge and Data Engineering*, **7**(4), 609–622.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., and Yanker, P. (1995). Query by image and video content: The QBIC system. *IEEE Computer*, **28**(9), 23–32.

Holt, B. and Hartwick, L. (1994). Visual image retrieval for applications in art and art history. In W. Niblack and R. C. Jain, editors, *SPIE Proceedings of Storage and Retrieval for Image Video Databases II*, volume 2185, pages 70–81. SPIE.

Lam, C. P., Wu, J. K., and Mehtre, B. (1995). STAR – a system for trademark archival and retrieval. In *Proceedings of the Second Asian Conference on Computer Vision*, pages III214–217. IEEE.

Lee, S. and Hsu, F. (1990). 2D C-string: A new spatial knowledge representation for image database systems. *Pattern Recognition*, **23**(10), 1077–1087.

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

Rouvray, D. H. and Balaban, A. T. (1979). Chemical applications of graph theory. In R. J. Wilson and L. W. Beineke, editors, *Applications of Graph Theory*, pages 177–221. Academic Press.

Shearer, K., Venkatesh, S., and Kieronska, D. (1997a). The visitors guide: a simple video reuse application. *International Journal of Pattern Recognition and Artificial Intelligence*, **11**(2), 275–301.

Shearer, K., Bunke, H., Venkatesh, S., and Kieronska, D. (1998a). Efficient graph matching for video indexing. *Computing*, **12**, 53–62.

Shearer, K., Venkatesh, S., and Bunke, H. (1998b). An efficient least common subgraph algorithm for video indexing. In *Proceedings of the International Conference on Pattern Recognition*, volume II, pages 1241–1243. IAPR, IEEE Computer Society.

Shearer, K. R., Kieronska, D., and Venkatesh, S. (1997b). Resequencing video using spatial indexing. *Journal of Visual Languages and Computing*, **8**, 193–214.

Shearer, K. R., Venkatesh, S., and Kieronska, D. (1997c). Spatial indexing for video databases. *Journal of Visual Communication and Image Representation*, **7**(4), 325–335.

Shotton, D. M., Rodriguez, A., Guil, N., and Trelles, O. (2000). Analysis and content–based querying of biological microscopy videos. In *Proceedings of the 15th International Conference on Pattern Recognition*. IAPR, IAPR.

Smith, M. A. and Kanade, T. (1998). Video skimming and characterization through the combination of image and understanding techniques. In *IEEE International Workshop on Content Based Access of Image and Video Databases*, pages 61–70.

Ullman, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, **23**(1), 31–42.