Video Indexing and Similarity Retrieval by Largest Common Subgraph Detection using Decision Trees

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

IDIAP–RR 00-15

April 2000

to appear in
Pattern Recognition

[a] IDIAP
[b] Institut für Informatik und Angewandte Mathematik, Universität Bern
[c] Curtin University of Technology

IDIAP Research Report 00-15

Video Indexing and Similarity Retrieval by Largest Common Subgraph Detection using Decision Trees

Kim Shearer Horst Bunke Svetha Venkatesh

April 2000

to appear in
Pattern Recognition

Abstract. While the largest common subgraph (LCSG) between a query and a database of models can provide an elegant and intuitive measure of similarity for many applications, it is computationally expensive to compute.

Recently developed algorithms for subgraph isomorphism detection take advantage of prior knowledge of a database of models to improve the speed of online matching. This paper presents a new algorithm based on similar principles to solve the largest common subgraph problem. The new algorithm significantly reduces the computational complexity of detection of the LCSG between a know database of models, and a query given online.

1 Introduction

With the advent of large online databases of images and video, it is becoming increasingly important to address the issues of effective indexing and retrieval. The majority of existing techniques in the area of image and video indexing can be broadly categorised into low level and high level techniques. Low level techniques use attributes such as colour and texture measures to encode an image, and perform image similarity retrieval by comparing vectors of such attributes.[6, 12, 15, 16, 25, 28] High level techniques use semantic information about the meaning of the pictures to describe an image, and therefore require intensive manual annotation.[1, 2, 7, 8, 11, 13, 14, 27] An additional method of image indexing is by qualitative spatial relationships between key objects. This form of index has been successfully applied to indexing and retrieval of image data.[4, 33, 36] Recent work by Shearer, Venkatesh[30, 31] has extended the range of this representation to encoding of video data. With the object information proposed for inclusion in the MPEG–4 standard, such indexes will be simpler to implement. It should be noted that none of these indexes provide a comprehensive retrieval method, but may be considered complementary methods, each being one component of a retrieval toolkit.

When images or video are encoded using spatial information, searches for exact pictorial or sub-picture matches may be performed using a compact encoding such as 2D–strings.[9, 10, 18] Similarity retrieval, however, is best expressed as inexact isomorphism detection between graphs representing two images.[17, 19] Images are encoded as graphs by representing each object in an image as a vertex in the graph, and placing an edge, labelled with the spatial relationship between the two corresponding objects, between each pair of vertices. Note that this is only one plausible representation of images using graphs. Other representations are possible, and in some cases will be preferable. Indeed any relational structure represented as a graph could make use of the algorithms discussed in this paper.

Inexact isomorphism detection between two graphs can be performed using one of two measures of similarity; edit distance or largest common subgraph. While there are well known algorithms to solve these problems, these algorithms are exponential in time complexity in the general case. This is a disadvantage when retrieval is likely to involve browsing of a large database, then progressive refinement.

Recently, new graph isomorphism algorithms have been developed by Messmer and Bunke,[21, 22] which use *a priori* knowledge of the database of model graphs to build an efficient index off-line. This index takes advantage of similarity between models to reduce execution time. Messmer and Bunke propose algorithms to solve the subgraph isomorphism problem, and to solve the inexact subgraph isomorphism problem using an edit distance measure. While subgraph isomorphism is often used as a measure of similarity between images, the edit distance measure is not suitable for image and video indexing by spatial relationships.

The major difficulty when applying edit distance methods to the image similarity problem, is that there is no clear interpretation for the edit operations. Deletion of a vertex implies the exclusion of an object from the matching part of two graphs, altering the label of an edge represents altering the relationships of two objects, there is no meaningful comparison for these two operations. This problem means that any similarity measure based on graph edit distance will return a value with little or no physical significance. Inexact isomorphism algorithms based on an edit distance measure also suffer from bias when used to compare an input against multiple model graphs. By the nature of edit distance algorithms, if the input graph is smaller than the models, smaller model graphs are more likely to be chosen as similar.

A more appropriate measure of image similarity is the largest common subgraph between the graphs representing the images. Largest common subgraph is a simple and intuitive measure of image similarity. The largest common subgraph between two graphs G_1 and G_2, encoding two images I_1 and I_2, represents the largest collection of objects found in I_1 and I_2 that maintain the same relationship to each other in both images. The usual algorithm for determining largest common subgraphs is the maximal clique detection method proposed by Levi.[20]

The maximal clique detection algorithm is efficient in the best case, but in the worst case requires $O((nm)^n)$ time, where n is the number of vertices in the input graph and m the number of vertices in the model, for a complete graph with all vertices sharing the same label. This high computational complexity makes it difficult to apply indices which are based on spatial relationships to large databases.

In this paper we describe a new algorithm for largest common subgraph detection. The algorithm has been developed from an algorithm originally proposed by Messmer and Bunke.[23,24] As with the original algorithm, this algorithm uses a preprocessing step, applied to the models contained in the database, to provide rapid classification at run time. The proposed algorithm is considerably more efficient in time than any previous algorithm for largest common subgraph detection, however the space complexity of the algorithm somewhat restricts its application. If the difference in time complexity is considered, from $O(L(nm)^n)$ for the usual algorithm for matching an input of size n, to a database of L models of size m, to the new algorithms complexity of $O(2^n n^3)$, there is room to perform space saving operations while still significantly out performing typical algorithms. One example of space saving is to depth limit the decision tree, such that searching is halted when a certain depth is reached, and matching between the few graphs left to separate is completed using a algorithm such as Ullman's,[35] instantiated from the matching already performed.

While largest common subgraph has been applied mostly to image similarity retrieval, the application of these new algorithms to indexing by spatial relationships can also take advantage of the high degree of common structure expected in video databases. In this paper we treat a video as a simple sequence of images. Even with this straight forward treatment it is possible to provide a similarity retrieval scheme that is extremely efficient, due to the high degree of common structure encountered in between frames in video. Many frames of a video will be classified by one element of the classification structure used. This removes the impediment of slow similarity retrieval for indices of this type. Furthermore, the algorithms may be applied to labelled, attributed, directed or undirected graphs. Such graphs have a great deal of expressive power, and may be applied to other encodings of image and video information, or other data sets of relational structures.

This paper begins by providing definitions for the key graph related concepts. Following sections will then briefly describe the encoding used for image and video information. The precursor to the new algorithm is then explained, followed by an explanation of the new algorithm. The results section compares the new algorithm with other available algorithms and examines the space complexity over an image database.

1.1 Definitions

Definition 1 *A* graph *is a 4-tuple $G = (V, E, \mu, \nu)$, where*

- *V is a set of vertices*
- *$E \subseteq V \times V$ is the set of edges*
- *$\mu : V \to L_V$ is a function assigning labels to vertices*
- *$\nu : E \to L_E$ is a function assigning labels to the edges*

Definition 2 *Given a graph $G = (V, E, \mu, \nu)$, a* subgraph *of G is a graph $S = (V_S, E_S, \mu_S, \nu_S)$ such that*

- $V_S \subseteq V$
- $E_S = E \cap (V_S \times V_S)$
- *μ_s and ν_S are the restrictions of μ and ν to V_S and E_S respectively, i.e.*

$$\mu_s(v) = \begin{cases} \mu(v) & \text{if } v \in V_S \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \nu_s(e) = \begin{cases} \nu(e) & \text{if } e \in E_S \\ \text{undefined} & \text{otherwise} \end{cases}$$

Table 1:

The notation $S \subseteq G$ is used to indicate that S is a subgraph of G.

Definition 3 *A bijective function $f : V \to V'$ is a graph isomorphism from a graph $G = (V, E, \mu, \nu)$ to a graph $G' = (V', E', \mu', \nu')$ if*

1. $\mu(v) = \mu'(f(v))\ \forall v \in V$
2. *For any edge $e = (v_1, v_2) \in E$ there exists an edge $e' = (f(v_1), f(v_2)) \in E'$ such that $\nu(e) = \nu(e')$, and for any $e' = ({v_1}', {v_2}') \in E'$ there exists an edge $e = (f^{-1}({v_1}'), f^{-1}({v_2}')) \in E$ such that $\nu(e') = \nu(e)$*

Definition 4 *An injective function $f : V \to V'$ is a* subgraph isomorphism *from G to G' if there exists a subgraph $S \subseteq G'$ such that f is a graph isomorphism from G to S*

Note that finding a subgraph isomorphism from G to G' implies finding a subgraph of G' isomorphic to the whole of G. This distinction becomes important in later discussion.

Definition 5 *S is a largest common subgraph of two graphs G and G', where $S \subseteq G$ and $S \subseteq G'$, iff $\forall S' : S' \subseteq G \wedge S' \subseteq G' \implies |S'| \leq |S|$*

There may or may not be a unique largest common subgraph for any two graphs G and G'.

2 Image indexing and retrieval

Before describing algorithms for retrieval it is necessary to discuss the underlying encoding of video. The encoding presented here is intended as a working example of how graph isomorphism may be used in image and video indexing and retrieval, but it is by no means the only such representation possible. Numerous methods have been proposed for indexing and retrieval of image and video information. These methods vary in the amount of preprocessing required and the type of information available for queries. The index used in this paper is based in spatial relationships between objects.

Spatial relationships between objects in a picture may be described in a number of ways. The most precise description is that proposed initially by Allen[3] for qualitative temporal reasoning. Allen gives the 13 possible relationships between two intervals along an axis (see table 1). This may be extended to two dimensions, by assigning one relation along each axis, giving a total of 169 possible relationships between two objects. The two axis are called either the u and v axis, or the x and y axis. By convention u or x refers to the horizontal axis, with v or y referring to the vertical axis.

A less precise description, based upon the same concepts, may be derived using the relationship categories proposed by Lee, Yang and Chen.[17] The 169 possible spatial relationships in two dimensions may be partitioned into five categories, defined by characteristics of the relationships. The categories are:

Definition 6

1. *Disjoint* – the two objects a and b do not touch or overlap, that is there is a *less than* $<$ operator along at least one axis.
2. *Meets* – the two objects a and b touch but do not overlap, thus they have a *meets* $|$ relationship along one axis and a non-overlapping relationship along the other.

3. *Contains* – object a contains object b, that is object a *contains* %, *begins* [or *ends*] object b along both axes.

4. *Belongs to* – object b contains object a, that is object b *contains* %, *begins* [or *ends*] object a along both axes.

5. *Overlaps* – the two objects do not fall into any of the above categories.

These category relationships are rotation and scaling invariant, making them useful as an approximate matching strategy.

General matching schemes based on spatial relationships define three levels of pictorial matching. These levels define the form of correspondence that is required between the spatial relationships of two objects, which appear in two pictures, for the pictures to be considered a matching pair. The matching scheme used in this work is adopted from the B–string notation,[17] which is the underlying notation used for storage of spatial relationships. B–strings provide three types of approximate matching, referred to as type–0, type–1 and type–2. Type–2 matching is the most exact, requiring that two objects a and b, present in two pictures P and Q, must have the same spatial relationship along both axes. Thus they must have the same pair, of the 169 possible pairs, of interval relationships. Type–0 matching is the least exact and uses relationship categories, requiring only that for a and b, the spatial relationships fall in the same relationship category. The final type of matching, type–1, requires that the a and b are a type–0 match, with the additional condition that the objects have the same orthogonal relationships. Insisting on matching orthogonal relationships in addition to a type–0 match restricts type–1 matching to a similar rotational position.

Two pictures P and Q are said to be a type–n match if:

1. For each object $o_i \in P$ there exists an $o_j \in Q$ such that $o_i \equiv o_j$, and for each object $o_j \in Q$ there exists an $o_i \in P$ such that $o_j \equiv o_i$

2. For all object pairs o_i, o_j where $o_i \in P$ and $o_j \in Q$, o_i and o_j are a type–n match

In many cases two pictures may not be complete matches, or even share the same object set, but we may be interested in partial matches. The search for partial matches, or subpicture matches is referred to as similarity retrieval. A subpicture match between P and Q is defined as:

1. For all objects $o_i \in Q'$, where $Q' \subset Q$, there exists an object $o_j \in P'$, where $P' \subset P$, such that $o_i \equiv o_j$

2. For all objects $o_j \in P'$, where $P' \subset P$, there exists an object $o_i \in Q'$, where $Q' \subset Q$, such that $o_j \equiv o_i$

3. For all object pairs o_i, o_j where $o_i \in P'$ and $o_j \in' Q$, o_i and o_j are a type–n match

Thus a subpicture match occurs when a subset of the objects in a picture P are found in a picture Q, with the spatial relationships being a type–n match.

When searching a video database, if there are no complete picture matches, we will be interested in the largest subpicture match that may be found. The classic methods for solving this problem cast the problem as subgraph isomorphism detection. When applied to image or video retrieval, the largest common subgraph method will find the largest subpicture in common between a query image and a database of images. Whether this is the desired retrieval result depends upon a number of factors.

The largest common picture may not contain one or more key objects which are considered essential by the user, or may contain additional objects or have general characteristics which are unsuitable. For this reason a ranked list of the closest matching images is generally returned. Modification of the

Figure 1:

query, in conjunction with selection of a more exact, or less exact matching type may then be utilised to refine the retrieval set.

Variation of type of matching employed allows either

- restriction of the retrieval set by increased exactitude in matching
- expansion of the retrieval set by reduced exactitude in matching

This, combined with small alterations to the query, allows a rapid focus on the desired results during retrieval. The qualitative nature of the relationships between objects and the less exact matching types allow noise to be accommodated in the browsing and query refinement process. Retrieval for this method is well defined and exact, the image in the database with the greatest number of objects in relationships which match the query will be the first model retrieved. How useful the retrieval method is depends on the form of retrieval desired and the flexibility of the three matching types. A discussion of usefulness and accuracy for a specific video retrieval application can be found in earlier work by the authors.[29, 30]

3 Encoding videos

Video may be encoded using techniques from image database work, adapted to the task of capturing motion in video.[5, 31] In order to keep the representation efficient in space usage, only changes in the video should be represented, rather than duplicating information between frames. This can be achieved by representing the initial frame of a video sequence using 2D strings, then encoding only the changes in spatial relationships between objects for the rest of the sequence. The earlier work performed by this group[31] encodes changing relationships such that matching can be performed directly from the abbreviated notation, without expansion of the notation. This encoding of video using spatial relationships applies graph algorithms to solve similarity retrieval.

In order to use a graph algorithm to solve a particular problem, it is first necessary to encode the operands of the problem as graphs. In this case the operands are either digital pictures or frames from a video. These frames are indexed by the spatial relationships of key objects. The logical graph encoding used for such information is that graph vertices represent objects, with edges labelled by the relationships between the objects. For the task of finding graph isomorphisms this leads to a complete labelled graph, as deduction of relationships at run time would be prohibitively time consuming. In practice, edges are labelled with the relationship category (definition 6) of the relationships between the two objects, with the actual relationship along each axis used as attributes of the edge. This representation is more efficient as all matching types require that two objects are at least a type–0 match, that is they have the same relationship category. Matching can therefore be performed by testing the edge label, or relationship category, and proceeding to the attributes if type–1 or type–2 matching is required and the labels are equivalent.

Figure 1 shows two pictures and the graphs which represent them. There is a clear similarity between the two pictures, the most obvious being between the subpictures composed of objects A, B and C. In the context of qualitative reasoning, the object B does not move with respect to A and C, as the relationships between them do not change. An examination of the two graphs reveals that if we remove the vertex for object D, and all arcs leading to or from that vertex, then the remaining parts of the graphs are identical. That is, the remaining vertices and arcs are a subgraph of the first graph isomorphic to a subgraph of the second. In the context of similarity retrieval in pictorial databases, we are interested in the elements of the database which share the *largest isomorphic subgraph with the query input.* This subgraph is referred to as the largest common subgraph, and is similar to the longest common subsequence problem for text strings.

Figure 2:

This encoding of images as graph enables one image to be compared to another and the largest similar part detected. The algorithms presented in the next section are applicable to similarity retrieval for both image and video databases. While they still perform matching on a single input query image basis, the compilation of multiple model images into a database explicitly takes advantage of the similarity between frames within a video. It is in part this similarity that leads to the efficiency of the proposed algorithms.

4 Decision tree algorithms

The decision tree based algorithm detects graph and subgraph isomorphisms from the input graph to the model graphs. That is, it finds subgraphs of the model graphs that are isomorphic to the input graph. This type of isomorphism detection is important as it is required for query by pictorial example, which is a common form of image database query. When query by pictorial example is used as the retrieval method, the input is an iconic sketch containing key objects. This iconic sketch is translated to a graph for which the vertex set generally represents a subset of the objects in the images retrieved.

The decision tree algorithm is based on a decision tree constructed using the adjacency matrix representation for the model graphs. A graph G may be represented by a matrix M, called the adjacency matrix, where the elements of M are assigned values

$$m_{ij} = \begin{cases} \mu(v_i) & \text{if } i = j \\ \nu((v_i, v_j)) & \text{if } i \neq j \end{cases} \tag{1}$$

This places the object labels down the diagonal of the matrix, and the labels of the edges in elements corresponding to the vertices they link. Figure 2 shows a graph with an adjacency matrix that can be used to represent it.

The property of adjacency matrices which leads to the decision tree algorithm is their behaviour under permutation. A permutation matrix is defined as

Definition 7 *An $n \times n$ matrix $P = (p_{ij})$ is called a* permutation matrix *if*

1. $p_{ij} \in 0, 1$ *for* $1 \leq i \leq n, 1 \leq j \leq n$
2. $\sum_{i=1}^{n} p_{ij} = 1$ *for* $1 \leq j \leq n$
3. $\sum_{j=1}^{n} p_{ij} = 1$ *for* $1 \leq i \leq n$

If element $p_{ij} = 1$ in a permutation matrix P, then applying the transformation

$$M' = PMP^T$$

will cause the j^{th} vertex in adjacency matrix M to become the i^{th} vertex in M'. One property of adjacency matrices is that given a graph G that is represented by an adjacency matrix M, then any matrix M', where $M' = PMP^T$ and P is a permutation matrix, is also an adjacency matrix representing G. Graph isomorphism detection between two graphs M_1 and M_2 can therefore be recast as the task of finding a permutation matrix P such that

$$M_2 = PM_1P^T$$

where M_1 and M_2 are the two adjacency matrices representing graphs G_1 and G_2 respectively.

Figure 3:

Figure 4:

In order to use this property to increase the speed of isomorphism detection, a decision tree is constructed from the adjacency matrices which represent the model graphs. This tree is created and navigated using *row column elements* of the adjacency matrices. Figure 3 shows the adjacency matrix from figure 2 broken into its row column elements. Each row column element r_i contains one vertex label v_i and the labels of all edges between v_i and vertices $v_1 \dots v_{i-1}$. The decision tree for a graph G begins with an unlabelled root node, which has as many descendants as there are distinct vertex labels. Each of these initial branches is labelled with a single vertex label, these represent the initial one element row column elements of each of the possible adjacency matrices for G. An example of this is given in Figure 4 which shows the six adjacency matrices which can represent the graph in figure 3, and the resulting decision tree. There are only two unique labels in the graph so there are only two descendants from the root node. These immediate descendants of the root have one descendent for each of the three element row column elements that follow them in one of the adjacency matrices. Figure 4 shows the decision tree generated from the six possible permutations of the example graph.

Further graphs, representing other images, can be added to the tree incrementally. For each additional adjacency matrix A, representing a graph G, to be added, the following algorithm is followed. Beginning with the root node of the tree, and the first row column element:

1. Test the next row column element of A against the labels on the branches descending from the current node.
2. If the last row column element of A has been reached, place a marker at the current node of the decision tree to say that this node represents an isomorphism for the model graph G.
3. If there is a matching branch label continue this procedure at the node reached along the matching branch, with the next row column element.
4. If there is no matching branch, create a new branch, labelled with the current row column element, and descend that branch to the next node. Continue from the new node with the next row column element.

This algorithm leads to the addition of only those branches which are not already represented in the tree. The higher the degree of similarity between models, the smaller the increase in tree size per model added.

The algorithm used to detect isomorphisms between an input graph G_I and the model graphs encoded in the decision tree is similar to the procedure used to add new adjacency matrices to the decision tree. Given an input graph G_I to test for subgraph isomorphisms, the adjacency matrix M_I which represents G_I is used to navigate the decision tree. Beginning at the root with the one element row column element at the top left of M_I, the algorithm descends the branches matching each of the row column elements of M_I. When descending the tree there are two possible termination conditions for the algorithm.

1. If at any point there is no arc from the current node in the tree which has a label matching the next row column element, then there is no subgraph isomorphism.
2. If all row column elements of the graph G_I have been used and node n_i has been reached, then all models $G_{i_1} \dots G_{i_k}$, associated with node n_i have a subgraph isomorphic to the input graph.

In case 2, if the node n_i is a leaf then the models $G_{i_1} \dots G_{i_k}$ and input G_I are the same size and a graph isomorphism, rather than a subgraph isomorphism, has been found.

A row column element r_i contains one vertex label v_i and the edge labels for each edge which connects r_i to any vertex $v_j : j < i$. Thus each row column element encapsulates all edges between v_i and vertices which have already been classified. The process of matching a row column element and descending to a new node in the tree is an incremental addition of vertices to the classified graph.

There are numerous optimisations which may be made to this algorithm,[22] involving methods of reducing the number of nodes in the tree. The best space complexity that may be achieved without some form of pruning is

$$O(L \mid l_v \mid (1 + \mid l_e \mid^2)^n) \tag{2}$$

where l_v is the number of vertex labels used and l_e is the number of edge labels used in the models. This figure can be achieved while still representing all states in the graph. Further heuristics can be used to prevent generation of further states at strategic points.

While this size complexity is an impediment to the use of this algorithm in general graph matching problems, there are problem domains for which the number of object labels can be small. An example of this is in medical image databases such as chest X-ray data. In such areas there are typically only four or five significant objects, making the decision tree algorithm appropriate.

The are also a number of methods for pruning the decision tree. This first is breadth pruning, in which nodes within the decision tree may eliminated by sorting the row column elements of the adjacency graphs. Here the row column elements are ordered such that each vertex is connected to at least one vertex which appears in an earlier row column element. This allows a number of permutations to be eliminated. The second method is depth pruning. This method places an upper limit on the depth of the decision tree. At any point in the construction of the decision tree where the maximum depth is reached, no further branches are built, but models are collected at the terminal node. If classification of an input reaches this terminal node, then Ullman's algorithm[35] is then used to complete classification. The vertex mappings used to reach the terminal node may also be used to initialise Ullman's algorithm, such that only the unmapped vertices are considered.

The advantage of the decision tree algorithm is that it has computational complexity is polynomial in the number of vertices in the input. The time taken to compute graph and subgraph isomorphisms is therefore independent of both the size of the model graphs and the number of model graphs. For an image or video database that will contain a large number of images, and will generally be queried by iconic query, this computational complexity is a clear advantage. Comparing the computational complexity of this algorithm of $O(n^2)$ to that of the previous best algorithm $O(Lm^n n^2)$, shows how rapidly this algorithm can classify the input.

Apart from the disadvantage of the space complexity of this algorithm which restricts its application, the inexact isomorphism method used with this algorithm has space complexity exponentially greater than the base algorithm. In order to apply this extremely fast algorithm to image and video databases it was necessary to develop an algorithm for similarity retrieval which is useable. The next section describes the LCSG decision tree algorithm developed for application to image and video indexing and retrieval.

4.1 Decision tree based LCSG algorithm

The advantage of the decision tree based largest common subgraph (LCSG) algorithm is that its computational complexity is independent of the number of models, and the size of the models. The complexity is dependent only on the size of the input, which is advantageous when the models are expected to be larger than the input, as in the case of query by pictorial example. This complexity is due to the depth of decent through the decision tree being limited to the number of vertices in the input.

The largest common subgraph (LCSG) for a pair of graphs is generally expensive to compute, due in part to its non-monotonic nature. This means that any LCSG algorithm must employ some form of backtracking to guarantee finding the optimal solution. The existence of a very fast algorithm for

Figure 5:

Figure 6:

detection of subgraph isomorphisms presents the possibility of a new approach to the largest common subgraph problem.

The algorithm developed is based on the observation that the decision tree algorithm may terminate classification before all possible matches for row column elements are discovered. The decision tree algorithm terminates as soon as a row column element is found for which there is no matching branch from the current node. At this point there may, however, be row column elements further down the adjacency matrix for which a match does exist. Consider the adjacency matrix in figure 5(a). When we attempt to classify this using the tree in figure 4, the algorithm terminates at the second row column element as there are no matches. However, if we permute the matrix of figure 5(a) such that the second and third row column elements are interchanged, we then have the matrix of figure 5(b). This represents a graph isomorphic to that represented by 5(a), but the permuted matrix allows both the first and second row column elements to be classified against the tree in Figure 4. This gives us a description of a common subgraph of order two for the graphs represented by the decision tree and the input. Clearly this procedure can be applied to larger examples.

When this method is applied to descend as far as possible through the tree, the resulting adjacency matrix is partitioned into two distinct parts. Figure 6 shows a graph and its adjacency matrix, and the resulting matrix after classification with respect to the decision tree of figure 4. The final matrix in figure 6 is marked to distinguish between the two partitions. The lower right partition contains the row column elements which could not be matched, and the upper left partition contains those row column elements which were matched. The upper left partition forms a permuted matrix which describes a potential largest common subgraph of the input and at least one model graph. The graph represented is only a potential LCSG, as any one vertex when included in the set of mapped vertices, may prevent correct detection of the largest common subgraph.

This problem is depicted in figure 7. An adjacency matrix representing the example graph in figure 7(a) is given in figure 7(b). When this adjacency matrix is classified using the decision tree from figure 4, the initial descent through the tree terminates with the partitioned matrix in figure 7(c). This shows a common subgraph of size two, with two unmatched row column elements. Figure 7(d) shows the partitioned matrix that results if row column element one is permuted to the end of the matrix before classification. Here the matrix shows a common subgraph of size three. Including node 1 therefore prevents detection of the LCSG.

The process of descending the decision tree to the best possible depth, by mapping row column elements where a match exists and permuting to the end of the adjacency matrix when there is no match, is given in the pseudo-code in figure 8, between labels $\boxed{\mathbf{1}}$ and $\boxed{\mathbf{4}}$. The block between labels $\boxed{\mathbf{2}}$ and $\boxed{\mathbf{3}}$ performs the step of permuting rows to the end of the adjacency matrix (function *permuteOut*) until either a matching row column element is found, or there are no further rows to test.

In order to find the largest common subgraph it is necessary to perform backtracking. Once the initial descent has been performed, the final node reached is a candidate node for the LCSG. Backtracking is performed by taking the partitioned matrix for the candidate node, and permuting the last row column element of the matched partition to the bottom of the matrix. The dimension of the matrix is then reduced by one to prevent a repeated match. This is performed at label $\boxed{\mathbf{5}}$ in the pseudo–code. Classification is then resumed at the immediate ancestor of the candidate node. Figure 7 shows a simple example of this backtracking scheme. After the initial descent, which detects

Figure 7:

Figure 8:

Figure 9:

a common subgraph of size two (figure 7(c)), row column element 4 is permuted to the end of the adjacency matrix, giving the matrix in figure 7(e). This matrix offers no further matches. Row three will then be permuted to the end, also yielding no further matches. The backtracking algorithm then returns to the previous level of matching by popping previous environments off a stack. This gives the original matrix with the first row column element as the only match. The first row column element is then permuted to the end of the matrix, giving the matrix in figure 7(d). At this point the whole matrix has been matched, row column element 1 having been discarded, so no further matching is required. It can easily be seen that this algorithm will examine every possible subgraph.

Such a search scheme is highly inefficient in this naive form, so a pruning mechanism is introduced. The search tree may be pruned at any point at which the number of nodes which have been permuted to the lower partition, plus the order of the best common subgraph detected so far, is greater than or equal to the current matrix dimension. For example assume during a continuing classification, a candidate subgraph has been found of size 5, from an input of size 8. If the classification has reached the third branch from the root, then the dimension of the input matrix will have been reduced to 6, as two row column elements (r_1 and r_2) have been discarded. This means that the best possible subgraph size on this descent is 6, so that any more than one permutation during the descent will make it impossible to exceed the current best effort of size 5.

The best pruning is seen when the depth of the initial descent is a large fraction of the full depth of the tree. At the extreme, if the dimension of the input adjacency matrix is d, and the depth of the initial descent is $d-1$, then pruning will occur before the second row column element is permuted out on all branches under the left most branch from the root node. On descents from the second branch from the root node pruning will occur when the first permutation is required, and no further branches under the root node will be examined as it would not be possible to equal the already discovered common subgraph. Consideration of this limit reveals it to be a powerful pruning factor, as shown by the following computational complexity analysis.

4.2 Complexity of tree based LCSG

For this complexity analysis we will assume that an input graph G_I with n vertices is to be classified against a database of L models of with m vertices. The complexity is derived using the equivalence of maximal cliques in association graphs and largest common subgraphs, to provide worst case complexity.

For two graphs G and G', the association graph is a vertex labelled, undirected graph which can be created by the two step process:

1. For each correct vertex mapping from graph G to G', insert a vertex in the association graph. This vertex is labelled with the vertex mapping between the vertices of G and G'.
2. For each pair of vertices v_i and v_j in the association graph, insert the edge $< v_i v_j >$ if the vertices mapped from G have the same edge characteristics as the vertices they are mapped to in G'.

Figure 9 shows two labelled graphs and the association graph produced for them. The vertices are labelled with the letters a to d, edges are directed but unlabelled, and the vertices of the two graphs are uniquely numbered for identification in the mappings of the association graph. This the mapping $1-5$ indicates the vertex labelled a in figure 9(a) is mapped to the vertex labelled a in figure 9(b).

Each clique within the association graph represents a set of vertices which have the same mutual relationships in G and G'. That is, the cliques represent subgraphs common to G and G'. The maximal clique or cliques in an association graph thus represent the largest common subgraph, or subgraphs. In order to determine the number of possible common subgraphs between two graphs, it is sufficient to examine the characteristics of cliques in an unlabelled, undirected graph of the appropriate size. This is used in analysing the worst case computational complexity.

The best case complexity for the new algorithm is found when the input is an exact subgraph isomorphism of at least one of the model graphs. In this case the time complexity is $O(n^2)$, as the algorithm simply descends directly down the tree. When there are no subgraphs with number of vertices greater than one, the complexity is simply $O(n^4)$. That is the cost of a descent with only one possible match, for each vertex in the input.

The worst case time complexity for this algorithm will occur when the size of the largest common subgraph is approximately half the size of the input. There are two factors which determine this:

1. The maximum possible number of common subgraphs is greatest.
2. The pruning due to best previous result has less effect.

The clearest example of the pruning effect can be seen if we have already found a subgraph of size $n-1$, in which case we can prune the search space at any point where a second permutation becomes necessary. This implies that only the two initial branches from the root will be examined.

In order to determine the maximum possible number of subgraphs for a graph of a given size, the following result due to Turan[34] is used. During this theory and the rest of this section the term order will be used for the number of vertices in a graph.

Theorem 1 *Every graph G on n vertices with*

$$1 + \binom{n}{2} - t \cdot \frac{n-c-r}{2}$$

edges contains a clique of order $c+1$, where $n = tc + r$, $0 \leq r < c$. This is the best possible.

The only graph G' of size

$$\binom{n}{2} - t \cdot \frac{n-c-r}{2}$$

which does not have a clique order $c+1$ is the complete c–partite graph with r parts of order $t+1$ and $c-r$ parts of order t.

The graph G' gives an indication of the maximal number of order c cliques possible in a graph of order n of

$$N_c = \left(\frac{n-r}{c} + 1\right)^r \cdot \left(\frac{n-r}{c}\right)^{c-r} \tag{3}$$

When the extra edge is added to G' to give a clique of order $c+1$, this will create

$$N_{c+1} = 2 \cdot \left(\frac{n-r}{c} + 1\right)^{r-1} \cdot \left(\frac{n-r}{c}\right)^{c-r} \tag{4}$$

cliques of order $c+1$. This leaves the number of cliques of size c as

$$\begin{aligned} N_{c'} &= N_c - N_{c+1} && (5)\\ &= \left(\frac{n-r}{c} + 1\right)^{r-1} \left(\frac{n-r}{c}\right)^{c-r} \left[\left(\frac{n-r}{c} + 1\right) - 2\right] && (6)\\ &= \left(\frac{n-r}{c} - 1\right) \left(\frac{n-r}{c} + 1\right)^{r-1} \left(\frac{n-r}{c}\right)^{c-r} && (7) \end{aligned}$$

Empirical analysis reveals that the practical limit to the function in equation 3 is 2^n, or more precisely $2^{\frac{n}{1.8}}$. The function $N_{c'}$ is bounded by 2^{n-1}, as the leading factor approaches unity. We shall use $N_{c'} = 2^n$ for our analysis, it should be noted that this is a generous over estimate.

A single descent through the decision tree has time complexity

$$C_d = n^2 + (n - c) \cdot C_{perm} \tag{8}$$

The second factor in equation 8 is the number of permutations required $(n - c)$, multiplied by the complexity of a permutation C_{perm}. The maximum complexity of a permutation is n^2, the cost of exchanging all elements. Equation 8 is therefore

$$C_d = n^3 + (1 - c) \cdot n^2 \tag{9}$$

Now given the number of cliques determined in equations 4 and 7, we can provide an upper bound for the time complexity of the algorithm of

$$C = 2^n \cdot (n^3 + (1 - (c + 1))n^2) + 2^{n-1} \cdot (n^3 + (1 - c) \cdot n^2) \tag{10}$$

Equation 10 consists of the number of cliques of size $c + 1$ multiplied by the cost of each descent, added to the number of cliques of size c multiplied by the cost of each descent. It can easily be seen from the construction of the graph that every vertex must contribute to at least one clique of size c, so the complexity calculation includes all descents. Removing an edge to reduce the size of a clique, or produce more backtracking, will greatly reduce the number of available cliques, and also reduce the computational complexity. The polynomial factors in this equation have been retained as the values of n for which this algorithm is used may be small, often no more than four.

The time complexity of the new algorithm, $O(2^n n^3)$, compares favourably with the maximal clique finding algorithm which has a worst case of $O((nm)^n)$.

5 Results of Tests over a Video Database

The video database used in the experiments was drawn mainly from our campus guide database.[30, 31] These video clips depict a guide walking between various locations on the Curtin University of Technology Campus. In addition to the clips from the campus guide, there are a number of other clips of park and city scenes, and a small number of disparate clips of completely different types of scenes.

The clips used vary in length from 4 seconds to 20 seconds, and contain between 12 and 19 objects each. The shortest clip contains 71 changes to object relationships, while the longest clip has 402 changes. These may be regarded as typical figures for changes per second, although one clip contains 225 changes in 7 seconds. The higher frequency of changes in relationship reflects the higher number of key objects in the clip.

All times given in tables are in milliseconds, and are averaged over a number of executions for each query. The algorithms used for comparison in this section are Ullman's algorithm for exact graph isomorphism detection,[35] the A* algorithm for inexact graph isomorphism detection,[26] and the decomposition network algorithms of Messmer and Bunke.[24] There are two decomposition network algorithms used as comparison, one using an edit distance measure (DN ED) and another using the largest common subgraph (DN LCSG) as the distance measure.[32]

5.1 Decision tree

There is no doubt that the decision tree algorithm is an exceptionally fast method for the detection of sub-pictures of models that are isomorphic to the input. Table 2 shows the times required for the decision tree algorithm to search two thirds of the guide database, containing the longest clips, for isomorphisms with example frames. The algorithm performs classification much faster than either Ullman's algorithm or either of the decomposition network algorithms. The decision tree algorithm

Table 2:

Table 3:

not only has by far the least mean for execution time, but also by far the least standard deviation and maximum. This speed of execution does not come without cost, in the form of increased space requirement over the other algorithms. Whereas Ullman's algorithm, the A* algorithm and both decomposition network algorithms all require similar space, and can load the entire database, it was not possible to load all clips using the decision tree algorithm. This is mostly due to the number of objects in two particular clips. Due to the high space requirement, there are a number of experiments in this section which examine the characteristics of the decision tree algorithm in this area.

One area in which the decision tree algorithm has a marked advantage is when there is no match available. In this case the decision tree algorithm can classify the input in 6 milliseconds, whereas for the example in table 2 the network algorithm averaged 39 milliseconds. In fact the decision tree algorithm performs at its best when the network algorithm performs at its worst.

The disadvantage of the decision tree algorithm is its space requirement. While it was possible to build a complete decision tree for two thirds of the guide database in 20Mb of memory, it was not possible to build a decision tree for just one of the omitted sequences in 300Mb. The reason for this is the exponential dependence of the decision tree algorithm upon the number of distinct vertex labels in the graph. In the case of image and video databases each vertex represents a key object. Whereas 4 clips with 9 or 10 objects per frame require a total of 18135 nodes in a decision tree, adding just two frames of a sequence with 19 objects increases the requirement to 76913 nodes. This experiment is described in table 4 in this section.

In an attempt to combat the size problems of the decision tree algorithm the GUB toolkit[21] includes a depth pruned algorithm. This permits the user to specify a maximum depth for the decision tree, once this depth is reached no further nodes are added, models being collected into the leaf node. In run time isomorphism detection, if a node is reached that has been depth limited, then detection continues using Ullman's algorithm, initialised with the results from the partial classification. This may allow rapid elimination of many alternatives, while still executing in a tractable size. The results of creating decision trees of varying depths for example sequences are given in table 3. Here we give the number of nodes required and the average time taken to match example pictures for the example depths of the decision tree. This shows that although the time required to search the database increases quite rapidly, the total time is still far less than that taken by any other algorithm.

Table 4 gives an indication of factors influencing the growth of the decision tree. Using a pruning depth of 5, the number of nodes was found for a number of combinations of clips from the database. Initially a database containing only the clip *liblr* was constructed, then a selection of other clips was added. *liblr* is relatively short, being only slightly over 5 seconds in length, and containing 12 indexed objects. Other clips added were:

librl A clip showing the same background as *liblr*, but with the guide walking in the opposite direction.

bookrl A clip of similar length and number of objects to the previous clips, but showing a different location, therefore with a largely disjoint object set.

booklr A clips showing the same background as *bookrl*, but with the guide walking in the opposite direction.

way1dl A clip 4 times the length of the previous clips, containing two less objects, with the object set largely disjoint from the previous clips.

cafelr A clip of similar length to the shorter clips, but containing 19 indexed objects, also with a largely disjoint object set.

Table 4:

Table 5:

Here we see that the addition of a clip containing a similar object set (1–2) causes only a minor increase in the number of nodes, while introducing a clip of similar size, with a disjoint object set (1–4), almost doubles the number of nodes. The increase in size is less than double due to the guide and exit being common objects in these clips. The fact that building a decision tree is a deterministic process is displayed by the equal number of nodes at 3 and 5. When a further sequence which shares most objects is added at 6 we see that a moderate increase in nodes occurs, yet noticeably less than the increase caused by the first sequence with that object set.

The final two lines display the pronounced effect of including a clip with an increased number of objects. At 7 in table 4 a sequence four times the length of the previous sequences has been added. This sequence has only ten objects, and causes the smallest increase in nodes required of any addition. In contrast to this, at 8 we have added only the first two frames from a sequence containing 18 objects, yet the increase in nodes required is far greater than the total number of nodes required for all other clips.

This clearly displays the weakness of the decision tree algorithm. Given a limited number of object labels, this algorithm can be used to detect database pictures or video frames which contain a user query picture with minimal execution time. However the number of labelled objects need only increase slightly to make the expression of the problem too expensive in space requirement. The original decision tree algorithm is also restricted to exact matching, with no inexact isomorphism detection method available.

5.2 Results of the LCSG algorithm

Table 5 gives the results of four queries performed using a number of inexact isomorphism detection algorithms. The algorithms used in comparison are the A* with lookahead algorithm, the decomposition network algorithm using edit distance as a similarity measure (DN ED) and the decomposition network algorithm using largest common subgraph as a distance measure (DN LCSG).[32] The first two queries have exact solution in the database, while *wayq* and *libq* have increasing differences (edit distance 6 and 12 respectively). As expected both algorithms based on edit distance (A* and DN ED) show a large increase in execution time as the error increases. The LCSG based algorithms show a decrease in execution time for one inexact query, and a time similar to exact times for the other. In all cases the decision tree LCSG algorithm is much faster than any other, an is at least two orders of magnitude faster than the A* algorithm.

Table 6 gives the results of the A*, the DN ED and the two LCSG algorithms for approximate matches between a single model graph and an input graph. The model graph and the input graphs have ten nodes each, and have been constructed to cause worst case performance for the LCSG algorithms. In each case the edit distance network algorithm performs isomorphism detection in a similar time to the decision tree LCSG algorithm, and these are the two fastest algorithms. The decomposition LCSG algorithm is slightly slower than the inexact decomposition algorithm, as expected given the nature of the graphs while the A* algorithm is much slower even for a single graph.

When the test in table 6 is extended to multiple models graphs, the performance of the decision tree LCSG algorithm varies little from that for a single model graph. The results in table 7 are for queries against 11 model graphs, of similar size and structure to the examples used in table 6. This model set is once again constructed to produce the worst case performance from the LCSG algorithms.

Table 6:

Table 7:

The results show that the time taken for the edit distance network algorithm increases considerably more slowly than that taken for the A* algorithm. The table also shows that the DT LCSG algorithm is essentially unaffected by the additional graphs in the database, out performing all other algorithms and returning the same time for the queries from table 6. This is as predicted by the complexity analysis. Given that the data set is constructed to provide worst case performance for the LCSG algorithms this is good performance.

While the A* and inexact network, and the LCSG do perform different computations, the end results are used for a similar purpose. The DT LCSG algorithm is actually more efficient than all other algorithms even in its worst case. The choice between LCSG algorithms should depend mostly on the characteristics of the problem, which determine the space complexity of the DT LCSG algorithm. Given a model size which makes the decision tree LCSG algorithm possible, it will provide significantly faster similarity retrieval than any other algorithm.

Whether the edit distance or LCSG algorithm is most appropriate for a given application depends on a number of factors. Just as there are tasks for which LCSG is the preferred measure of graph similarity, there are also tasks for which edit distance is a better measure. Structural properties of the model graphs should also be considered, the preprocessed algorithms being at their best when there is much common structure in the model graphs.

6 Conclusion

This paper presents a new algorithm for detection of the largest common subgraph between two graphs. The algorithm is intended for use with a database of models of which there is prior knowledge, and provides rapid on line processing of input queries. The contributions of this paper are to examine the performance of the algorithm and its ancestors over video database data, and the presentation of the new algorithm for largest common subgraph detection.

The alternative algorithms studied are suitable for certain problems in image and video database retrieval, however each is limited in this application as discussed. The largest common subgraph algorithm provides suitable solutions for the task of similarity retrieval for images and video in large databases. The strength of the algorithm presented in this paper is its exceptional classification performance, as shown in the results over video database data.

The algorithm presented here makes largest common subgraph detection tractable for large databases of small model graphs. This is a problem that is becoming more common as complex data such as images, video and sound become widely available in large quantities. These types of data require complex relational descriptions, which have traditionally been slow to process for approximate matching. Given prior knowledge of the database of model graphs the algorithm presented here offers large improvements in classification time for input graphs.

References

[1] Shinji Abe, Yoshinobu Tonomura, and Hisashi Kasahara. Scene retrieval method for video database applications using temporal condition changes. In *International Workshop on Industrial Applications of Machine Intelligence and Vision*, pages 355–359. IEEE, April 1989.

[2] Sibel Adah, Selçuk Candan, Su-Shing Chen, Kutluhan Erol, and V S Subrahmanian. The advanced video information system: data structures and query processing. *Multimedia Systems*, 4:172–186, 1996.

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

[4] T. Arndt and S. Chang. An intelligent image database system. In *Proceedings of the IEEE Workshop on Visual Languages*, pages 177–182. IEEE, 1989.

[5] Timothy Arndt and Shi-Kuo Chang. 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, October 1989.

[6] Jonathan Ashley, Ron Barber, Myron Flickner, James Hafner, Denis Lee, Wayne Niblack, and Dragutin Petkovic. Automatic and semi-automatic methods for image annotation and retrieval in QBIC. *SPIE Proceedings of Storage and Retrieval for Image Video Databases III*, pages 24–35, 1995.

[7] R. Burke and A. Kass. Refining the universal indexing frame to support retrieval of tutorial stories. In *Indexing and Reuse in Multimedia Systems*, pages 1–11, Seattle, WA, August 1994. AAAI, AAAI. Workshop Notes.

[8] 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.

[9] S. Chang, Q. Shi, and C. Yan. 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, May 1987.

[10] Shi-kuo Chang, Erland 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.

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

[12] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[13] B. R. Gaines and M. LG Shaw. Concept maps indexing multimedia knowledge bases. In *Indexing and Reuse in Multimedia Systems*, pages 36–45, Seattle, WA, August 1994. AAAI, AAAI. Workshop Notes.

[14] Bonnie Holt and Laura Hartwick. Visual image retrieval for applications in art and art history. In Wayne Niblack and Ramesh C Jain, editors, *SPIE Proceedings of Storage and Retrieval for Image Video Databases II*, volume 2185, pages 70–81. SPIE, February 1994.

[15] Anil K Jain and A Vailaya. Image retrieval using color and shape. In *Proceedings of the Second Asian Conference on Computer Vision*, pages II529–533. IEEE, 1995.

[16] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: active contour models. In *Proceedings of the First International Conference on Computer Vision*, pages 259–269, 1987.

[17] 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.

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

[19] Suh-yin Lee, Man-kwan Shan, and Wei-pang Yang. Similarity retrieval of iconic image database. *Pattern Recognition*, 22(6):675–682, 1989.

[20] G Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–354, 1972.

[21] B T Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs.* PhD thesis, Institut fur Informatik und angewandte Mathematik, Universitat Bern, Switzerland, 1995.

[22] B T Messmer and H Bunke. Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In *Second Asian Conference on Computer Vision*, pages 151–155, 1995.

[23] B T Messmer and Horst Bunke. Error–correcting graph isomorphism using decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(6):721–742, September 1998.

[24] B T Messmer and Horst Bunke. A new algorithm for error–tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, May 1998.

[25] Akio Nagasaka and Yuzuru Tanaka. Automatic video indexing and full-video search for object appearences. In E Knuth and L M Wegner, editors, *Visual Database Systems, II*, number A-7 in IFIP Transactions, pages 113–127, Elsevier Science Publishers, Sara Burgerhartstraat 25, P.O. Box 211, 1000 AE Amsterdam, The Netherlands, September 1992. IFIP, Elsevier Science Publishers.

[26] N J Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, 1980.

[27] L. A. Rowe, J. S. Boreczky, and C. A. Eads. Indexing for user access to large video databases. In W. Niblack and R. C. Jain, editors, *SPIE Proceedings of Storage and Retrieval for Image Video Databases II*, volume 2185, pages 150–161, San Jose, CA, February 1994. IS&T and SPIE.

[28] Simone Santini and Ramesh Jain. Similarity matching. In *Proceedings of the Second Asian Conference on Computer Vision*, pages II544–548. IEEE, 1995.

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

[30] K R Shearer, D Kieronska, and S Venkatesh. Resequencing video using spatial indexing. *Journal of Visual Languages and Computing*, 8:193–214, 1997.

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

[32] Kim Shearer, Svetha Venkatesh, and Horst Bunke. 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, August 1998.

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

[34] P Turán. Eine extremalaufgabe aus der graphentheorie. *Mat. Fiz. Lapok*, 48:436–452, 1941.

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

[36] Ming-Chwen Yang. 2D B-string representation and access methods of image database. Master's thesis, National Chiao Tung University, Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, July 1990.

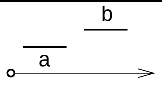
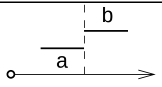
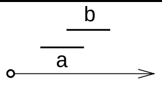
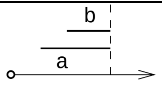
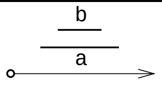
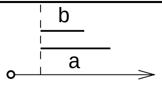
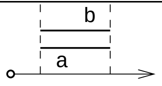
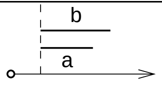
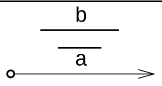
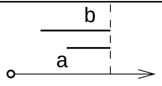
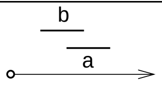
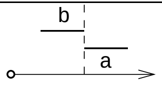
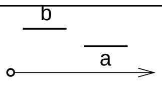
Relation	Symbol	Example	Relation	Symbol	Example
less than	`a<b`		*meets*	`a|b`	
overlaps	`a/b`		*ends*	`a]b`	
contains	`a%b`		*begins*	`a[b`	
equals	`a=b`		*begins inverse*	`a['b`	
contains inverse	`a%'b`		*ends inverse*	`a]'b`	
overlaps inverse	`a/'b`		*meets inverse*	`a|'b`	
less than inverse	`a<'b`				

Table 1: Possible interval relationships, due to Allen[3]

	Mean	Minimum	Maximum	δ
Ullman	393.2 μs	252 μs	607 μs	113.1 μs
*A**	617.1 μs	362 μs	861 μs	178.2 μs
DN ED	109.0 μs	81 μs	209 μs	36.2 μs
DN LCSG	67.4 μs	38 μs	100 μs	19.6 μs
DT LCSG	16.6 μs	6 μs	23 μs	6.5 μs

Table 2: General performance of the decision tree algorithm

Depth	Nodes used	Test 1	Test 2	Test 3	Test 4
11	59200	3 μs	3 μs	3 μs	3 μs
9	58900	11 μs	3 μs	7 μs	7 μs
8	56706	6 μs	6 μs	5 μs	5 μs
7	49130	6 μs	5 μs	6 μs	5 μs
6	34689	5 μs	4 μs	5 μs	4 μs
5	18135	4 μs	4 μs	5 μs	5 μs
4	6495	36 μs	17 μs	7 μs	6 μs
3	1499	46 μs	15 μs	11 μs	10 μs

Table 3: Effect of pruning depth

	Sequences	**Nodes**	**δ nodes**
1	*liblr*	6780	
2	*liblr librl*	8211	1431
3	*liblr librl bookrl*	13991	5780
4	*liblr bookrl*	12977	6197
5	*liblr bookrl librl*	13991	1014
6	*liblr librl bookrl booklr*	18135	4144
7	*liblr librl bookrl booklr way1dl*	19389	1254
8	*liblr librl bookrl booklr cafelr[0-1]*	76913	58788

Table 4: Effect of introducing sequences

Query	Error	A*	DN ED	DT LCSG	DN LCSG
liblr.10	0	10737μs	172μs	26 μs	78 μs
liblr.0	0	9073μs	164μs	22 μs	65 μs
wayq	6	9122μs	223μs	7 μs	46 μs
libq	12	35674μs	2851μs	14 μs	72 μs

Table 5: Performance of LCSG algorithm

Query	A*	DN ED	DT LCSG	DN LCSG
15.1	158 μs	23 μs	22 μs	34 μs
15.2	160 μs	22 μs	23 μs	35 μs
15.3	164 μs	22 μs	22 μs	34 μs

Table 6: Times for approximate match between two graphs

Query	A*	DN inexact	DT LCSG	DN LCSG
11.6	282 μs	30 μs	17 μs	38 μs
12.2	53 μs	15 μs	5 μs	14 μs
14.1	84 μs	18 μs	7 μs	16 μs
15.2	642 μs	34 μs	23 μs	45 μs
15.3	692 μs	36 μs	22 μs	46 μs
16.1	136 μs	20 μs	8 μs	20 μs

Table 7: Times for approximate match against 11 graphs

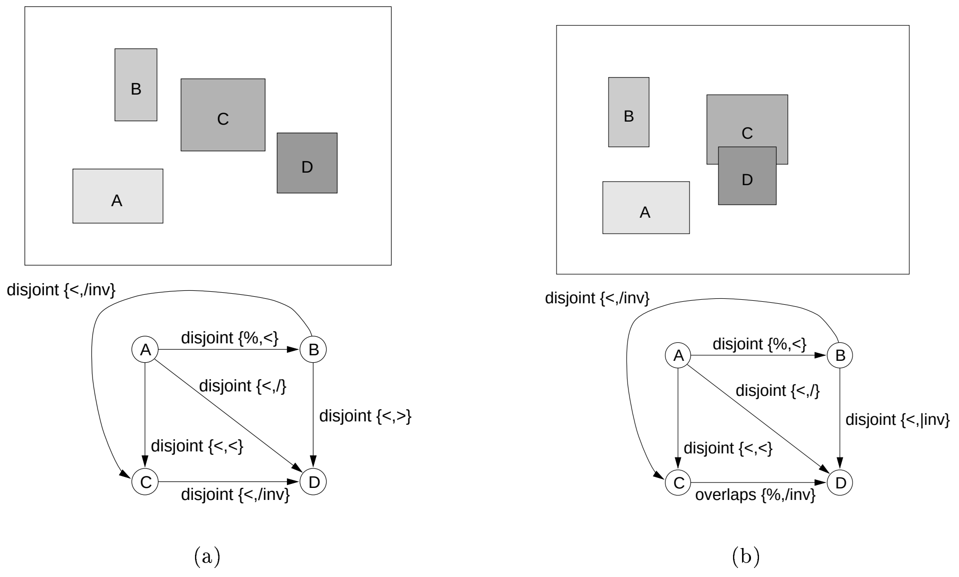


Figure 1: Two pictures with possible graph encodings

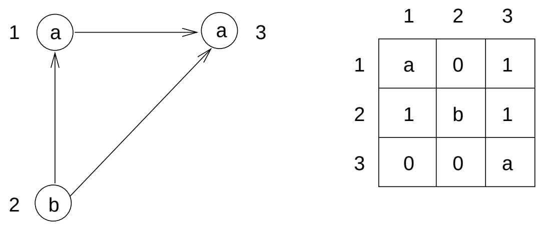


Figure 2: Graph with adjacency matrix

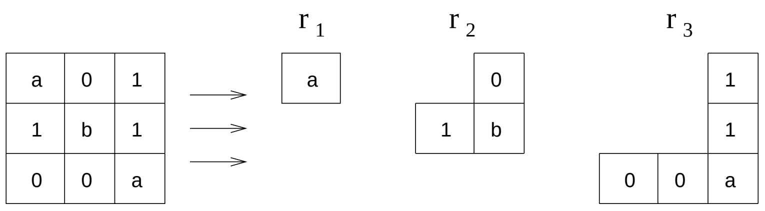


Figure 3: Row column elements of a matrix

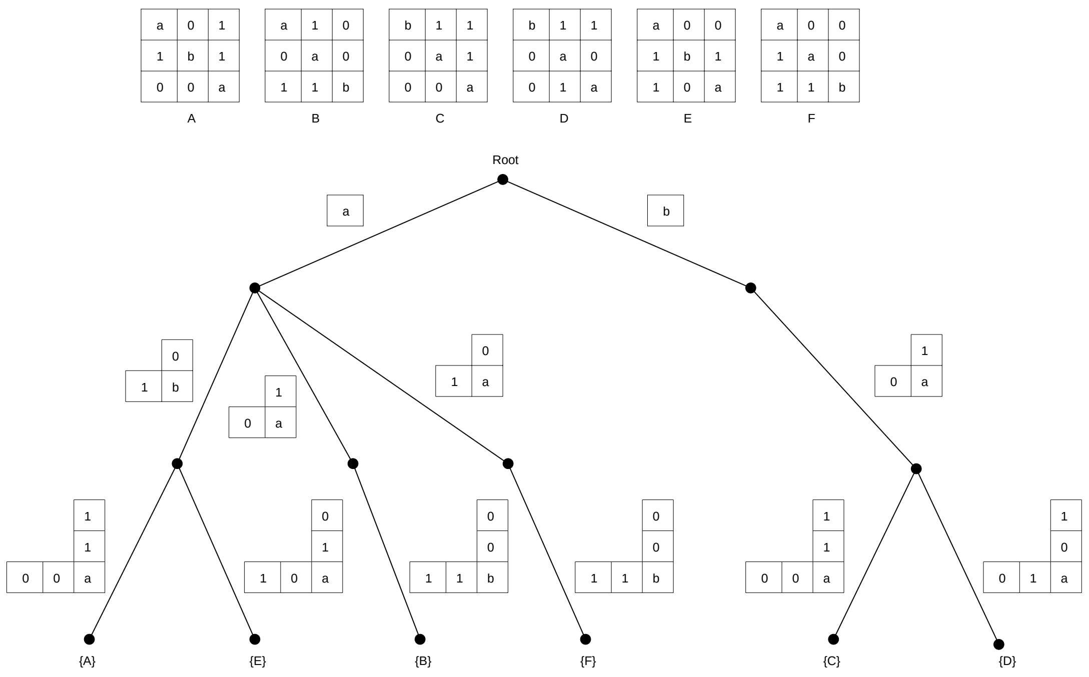


Figure 4: Decision tree for example graph

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

(a)

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

(b)

Figure 5: Input graph adjacency matrices

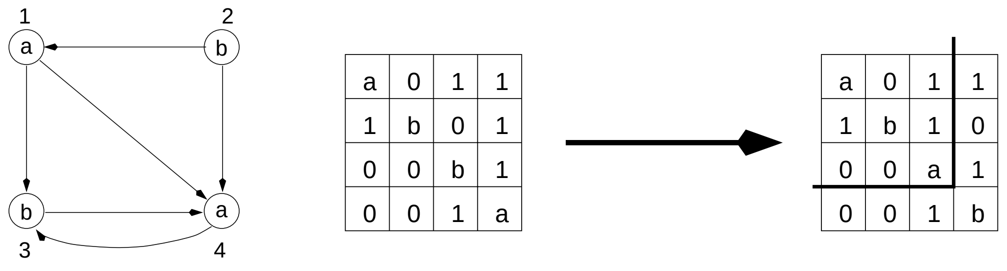
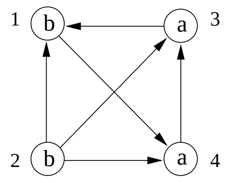


Figure 6: Partitions of the adjacency matrix



(a)

$$\begin{array}{c|cccc}
1 & b & 0 & 0 & 1 \\
2 & 1 & b & 1 & 1 \\
3 & 1 & 0 & a & 0 \\
4 & 0 & 0 & 1 & a
\end{array}$$

(b)

$$\begin{array}{c|cc|cc}
1 & b & 1 & 0 & 0 \\
4 & 0 & a & 1 & 0 \\
\cline{1-3}
3 & 1 & 0 & a & 0 \\
2 & 1 & 1 & 1 & b
\end{array}$$

(c)

$$\begin{array}{c|ccc|c}
2 & b & 1 & 1 & 1 \\
3 & 0 & a & 0 & 1 \\
4 & 0 & 1 & a & 0 \\
\cline{1-4}
1 & 0 & 0 & 1 & b
\end{array}$$

(d)

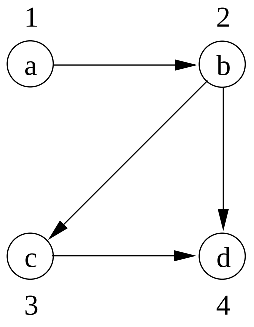
$$\begin{array}{c|cccc}
1 & b & 0 & 0 & 1 \\
3 & 1 & a & 0 & 0 \\
2 & 1 & 1 & b & 1 \\
4 & 0 & 1 & 0 & a
\end{array}$$

(e)

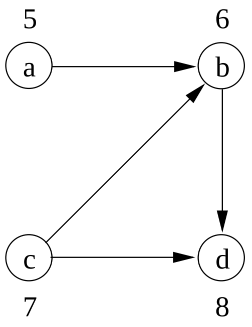
Figure 7: Permutation of matrices during classification

```
state = Root of decision tree; AM = Input adjacency matrix; depth=0
total_permutations=0; done=false
do
$\boxed{1}$ Dim=AM→Dimension
    while depth+total_permutations<Dim $\wedge$ best_depth+total_permutations$\leq$Dim
        do
            next=findNodeAtNextDepth (state,vertex_sequence[depth],AM)
            did_perm=false
            if next=null
                if depth+total_permutations<Dim−1 $\wedge$ best_depth+total_permutations<Dim
$\boxed{2}$                 AM→permuteOut(depth)
                    permuteVS(vertex_seq, depth)
                    total_permutations++
                    did_perm=true
$\boxed{3}$             else total_permutations++
        while did_perm
        if next$\neq$null
            pushEnvironment()
            state=setNextState(state, next)
            depth++
    if depth=best_depth
        addToLcsg(vertex_seq)
    else if depth>best_depth
        clearLcsg()
        addToLcsg(vertex_seq)
        best_depth=depth
$\boxed{4}$ if more environments to test
        popEnvironment()
        while more environments to test $\wedge$ AM→Dimension<best_depth
            popEnvironment()
        if AM→Dimension<best_depth
            done=true
        else
$\boxed{5}$         AM→permuteOut(depth)
            permuteVS(vertex_seq, depth)
            AM→Dimension−−
    else done=true
while $\neg$ done
```

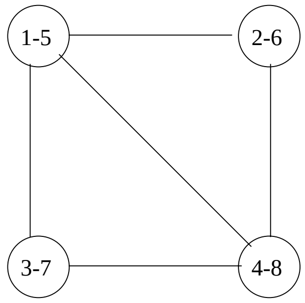
Figure 8: Pseudo–code for the decision tree LCSG algorithm



(a) Graph G



(b) Graph G'



(c) Association graph

Figure 9: Example of an association graph