Knowledge Discovery from Structural Data *

Diane J. Cook, Lawrence B. Holder and Surnjani Djoko
Department of Computer Science Engineering
University of Texas at Arlington
Email: cook, holder, djoko@cse.uta.edu

Keywords: machine discovery, data mining, data compression, inexact graph match, scene analysis, chemical analysis

*Supported by NASA grant NAS5-32337.

Abstract

Discovering repetitive substructure in a structural database improves the ability to interpret and compress the data. This paper describes the SUBDUE system that uses domain-independent and domain-dependent heuristics to find interesting and repetitive structures in structural data. This substructure discovery technique can be used to discover fuzzy concepts, compress the data description, and formulate hierarchical substructure definitions. Examples from the domains of scene analysis, chemical compound analysis, computer-aided design, and program analysis demonstrate the benefits of the discovery technique.

1 Introduction

The large amount of data collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns within the data. In response to this problem, a number of researchers have developed techniques for discovering concepts in databases [2, 5, 15]. These techniques work well for data expressed in a non-structural, attribute-value representation, and address issues of data relevance, missing data, noise and uncertainty, and utilization of domain knowledge. However, recent data acquisition projects are collecting structural data describing the relationships among the data objects. Correspondingly, there exists a need for techniques to analyze and discover concepts in structural databases.

One method for discovering knowledge in structural data is the identification of common substructures within the data. The motivation for this process is not merely to find substructures capable of compressing the data by abstracting instances of the substructure, but also to identify conceptually interesting substructures that enhance the interpretation of the data. Substructure discovery is the process of identifying concepts describing interesting and repetitive substructures within structural data. Once discovered, the substructure concept can be used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered concept. The discovered substructure concepts allow abstraction over detailed structure in the original data and provide new, relevant attributes for interpreting the data. Iteration of the substructure discovery and replacement process constructs a hierarchical description of the structural data in terms of the discovered substructures. This hierarchy provides varying levels of interpretation that can be accessed based on the goals of the data analysis.

We introduce a system that discovers interesting substructures in structural data called SUBDUE [7]. The SUBDUE system discovers substructures that compress that original database and represent interesting structural concepts in the data. By replacing previously-discovered substructures in the data, multiple passes of SUBDUE produce a hierarchical description of the structural regularities in the data. SUBDUE uses a computationally-bounded inexact graph match that identifies similar, but not identical, instances of a substructure and finds an approximate measure of closeness of two substructures when under computational constraints. In addition to the minimum description length principle, other background knowledge can be used by SUBDUE to guide the search towards more appropriate substructures.

The following sections describe the approach in detail. Section 2 describes other existing research in concept discovery. Section 3 introduces needed definitions and describes the basic substructure discovery process. Sections 4 and 5 describes two algorithms that are central to the graph-based discovery approach: the mdl encoding of input graphs and discovery substructures, and the inexact graph match procedure. Section 6 evaluates the system with the domain-independent heuristics using data from the domains of scene analysis, chemical analysis and CAD circuits. In Section 7 we describe the process for constructing a hierarchical description of the database from multiple discovery iterations, and in Section 8 we discuss methods of enhancing SUBDUE's performance using domain-dependent knowledge. Section 9 concludes with directions for future work.

2 Related Work

Several approaches to substructure discovery have been developed. Winston's ARCH program [24] discovers substructures in order to deepen the hierarchical description of a scene and to group objects into more general concepts. The ARCH program searches for two types of substructure in the blocks-world domain. The first type involves a sequence of objects connected by a chain of similar relations. The second type involves a set of objects each having a similar relationship to some "grouping" object. The main difference between the substructure discovery procedures used by the ARCH program and SUBDUE is that the ARCH program is designed specifically for the blocks-world domain. For instance, the sequence discovery method looks for `supported-by` and `in-front-of` relations only. SUBDUE's substructure discovery method is domain independent, although the inclusion of domain-specific knowledge would improve SUBDUE's performance.

Motivated by the need to construct a knowledge base of chemical structures, Levinson [11] developed a system for storing labeled graphs in which individual graphs are represented by the set of vertices in a universal graph. In addition, the individual graphs are maintained in a partial ordering defined by the `subgraph-of` relation, which improves the performance of graph comparisons. The universal graph representation provides a method for compressing the set of graphs stored in the knowledge base. Subgraphs of the universal graph used by several individual graphs suggest common substructure in the individual graphs. One difference between the two approaches is that Levinson's system is designed to incrementally process smaller individual graphs; whereas, SUBDUE processes larger graphs all at once. Also, Levinson's system discovers common substructure only as an indirect result of the universal graph construction; whereas, SUBDUE's main goal is to discover and output substructure definitions that reduce the minimum description length encoding of the graph. Finally, the `subgraph-of` partial ordering used by Levinson's system is not included in SUBDUE, but maintaining this partial ordering would improve the performance of the graph matching procedure by pruning the number of possible matching graphs.

Segen [21] describes a system for storing graphs using a probabilistic graph model to represent subsets of the graph. Alternative models are evaluated based on a minimum description length measure of the information needed to represent the stored graphs using the model. In addition, Segen's system clusters the graphs into classes based on minimizing the description length of the graphs according to the entire clustering. Apart from the probabilistic representation, Segen's approach is similar to Levinson's system in that both methods take advantage of commonalities in the graphs to assist in graph storage and matching. The probabilistic graphs contain information for identifying common substructure in the exact graphs they represent. The portion of the probabilistic graph with high probability defines a substructure that appears frequently in the exact graphs. This notion was not emphasized in Segen's work, but provides an alternative method to substructure discovery by clustering subgraphs of the original input graphs. As with Levinson's approach, graphs are processed incrementally, and substructure is found across several graphs, not within a single graph as in SUBDUE.

The LABYRINTH system [22] extends the COBWEB incremental conceptual clustering system [5] to handle structured objects. LABYRINTH uses COBWEB to form hierarchical concepts of the individual objects in the domain based on their primitive attributes. Concepts of structured objects are formed in a similar manner using the individual objects as attributes. The resulting hierarchy represents a componential model of the structured objects. Because COBWEB's concepts are probabilistic, LABYRINTH produces probabilistic models of the structured objects, but with an added hierarchical organization. The upper-level components of the structured-object hierarchy produced by LABYRINTH represent substructures common to the examples. Therefore, although not the primary focus, LABYRINTH is discovering substructure, but in a more constrained context than the general graph representation used by SUBDUE. Both Segen's work and the work by Thompson and Langley are similar to SUBDUE in their use of a graph representation and techniques such as graph match and hierarchical graph formation. However, the purpose of the SUBDUE system is to discovery knowledge, while the purpose of these other systems is to store information. In addition, SUBDUE is distinct in using the minimum description length principle as a primary evaluation measure and in allowing the use of both domain-independent and domain-dependent heuristics.

Conklin et al. [3] have developed the I-MEM system for constructing an image hierarchy, similar to that of LABYRINTH, used for discovering common substructures in a set of images and for efficient retrieval of images similar to a given image. Images are expressed in terms of a set of relations defined by the user. Specific and general (conceptual) images are stored in the hierarchy based on a subsumption relation similar to Levinson's `subgraph-of` partial ordering. Image matching utilizes a transformational approach (similar to SUBDUE's inexact graph match) as a measure of image closeness. Conklin's work is very effective in the domain of chemistry, but has not been applied outside this domain. This work has not been extended to consider domain-independent techniques or to allow application to additional domains of interest.

As with the approaches of Segen and Levinson, I-MEM is designed to process individual images. Therefore, the general image concepts that appear higher in I-MEM's hierarchy will represent common substructures across several images. SUBDUE is designed to discover common substructures within a single image. SUBDUE can mimic the individual approach of these systems by processing a set of individual images as one disconnected graph. The substructures found will be common to the individual images. The hierarchy also represents a componential view of the images. This same view can be constructed by SUBDUE using multiple passes over the graph after replacing portions of the input graph with substructures discovered during previous passes. I-MEM has performed well in a simple chess domain and molecular chemistry domains [3]. However, I-MEM requires domain-specific relations for expressing images in order for the hierarchy to find relevant substructures and for image matching to be efficient. Again, maintaining the concepts (images, graphs) in a partially-ordered hierarchy improves the efficiency of matching and retrieval, and suggests a possible improvement to SUBDUE.

The CLIP system [25] for graph-based induction is more similar to SUBDUE than the previous systems. CLIP iteratively discovers patterns in graphs by expanding and combining patterns discovered in previous iterations. Patterns are grouped into views based on their collective ability to compress the original input graph. During each iteration CLIP uses existing views to contract the input graph and then considers adding to the views new patterns consisting of two vertices and an edge from the contracted graph. The compression of the new proposed views is estimated, and the best views (according to a given beam width) are retained for the next iteration.

CLIP discovers substructures (patterns) differently than SUBDUE. First, CLIP produces a set of substructures that collectively compress the input graph; whereas, SUBDUE produces only single substructures evaluated using the more principled minimum description length. CLIP has the ability to grow substructures agglomeratively (i.e., merging two substructures together); whereas, SUBDUE always produces new substructures using incremental growth along one new edge. CLIP initially estimates the compression value of new views based on the compression value of the parent view; whereas, SUBDUE performs an expensive exact measurement of compression for each new substructure. Finally, CLIP employs an efficient graph match based on graph identity, not graph isomorphism as in SUBDUE. Graph identity assumes an ordering over the incident edges of a vertex and does not consider all possible mappings when looking for occurrences of a pattern in an input graph. These differences in CLiP suggest possible enhancements to SUBDUE.

Research in pattern recognition has begun to investigate the use of graphs and graph grammars as an underlying representation for structural problems [20]. Many results in grammatical inference are applicable to constrained classes of graphs (e.g., trees) [6, 12]. The approach begins with a set of sample graphs and produces a generalized graph grammar capable of deriving the original sample graphs and many others. The production rules of this general grammar capture regularities (substructures) in the sample graphs. Jeltsch and Kreowski [9] describe an approach that begins with a maximally-specific grammar and iteratively identifies common subgraphs in the right-hand sides of the production rules. These common subgraphs are used to form new, more general production rules. Although their method does not address the underlying combinatorial nondeterminism, heuristic approaches could provide a feasible method for extracting substructures in the form of graph grammars. Furthermore, the graph grammar production-rule may provide a suitable representation for background knowledge during the substructure discovery process.

3 Substructure Discovery

The substructure discovery system represents structured data as a directed graph. Objects in the data map to vertices or small subgraphs in the graph, and relationships between objects map to directed or undirected edges in the graph. A *substructure* is a connected subgraph within the graphical representation. This graphical representation serves as input to the substructure discovery system. Figure 1 gives an example input database and its graphical representation. The objects in the figure (e.g., `T1`, `S1`, `R1`) become labeled vertices in the graph, and the relationships (e.g., `on(T1,S1)`, `shape(C1,circle)`) become labeled edges in the graph. The graphical representation of the substructure discovered by SUBDUE from this data is shown in Figure 2.

An *instance* of a substructure in an input graph is a set of vertices and edges from the input graph that match, graph theoretically, to the graphical definition of the substructure. For example, the discovered substructure from Figure 1 and its instances are shown in Figure 2. A *neighboring edge* of a substructure instance is an edge in the input graph that is not contained in the instance, but is connected to at least one vertex in the instance. For example, the first instance has one neighboring edge: `on(S1,R1)`. An *external connection* of an instance of a substructure is a neighboring edge of the instance that is connected to at least one vertex not contained in the instance. Therefore, `on(S1,R1)` is also the only external connection of the first instance in Figure 2.

The substructure discovery algorithm used by SUBDUE is a computationally constrained beam search. The algorithm begins with a list of substructures matching distinct vertices in the graph. Each iteration through the algorithm selects the best substructure and expands instances of these substructures by one neighboring edge in all possible ways. The new unique generated substructures become candidates for further expansion, though the number of candidate structures is always less than or equal to the beam width. The algorithm continues to search for the best substructure until all possible substructures have been considered or the amount of computation exceeds a given limit.

Typically, once the description length of an expanding substructure begins to increase, further expansion of the substructure will not yield a smaller description length. As a result, SUBDUE makes use of an optional pruning mechanism that eliminates substructure expansions from consideration when the description lengths for these expansions increases.

The choice of heuristics used to evaluate each substructure will significantly affect the results of the system. If domain-independent heuristics are employed, the discovery process may yield surprising results. However, the discovered substructures may not always be useful to the user. On

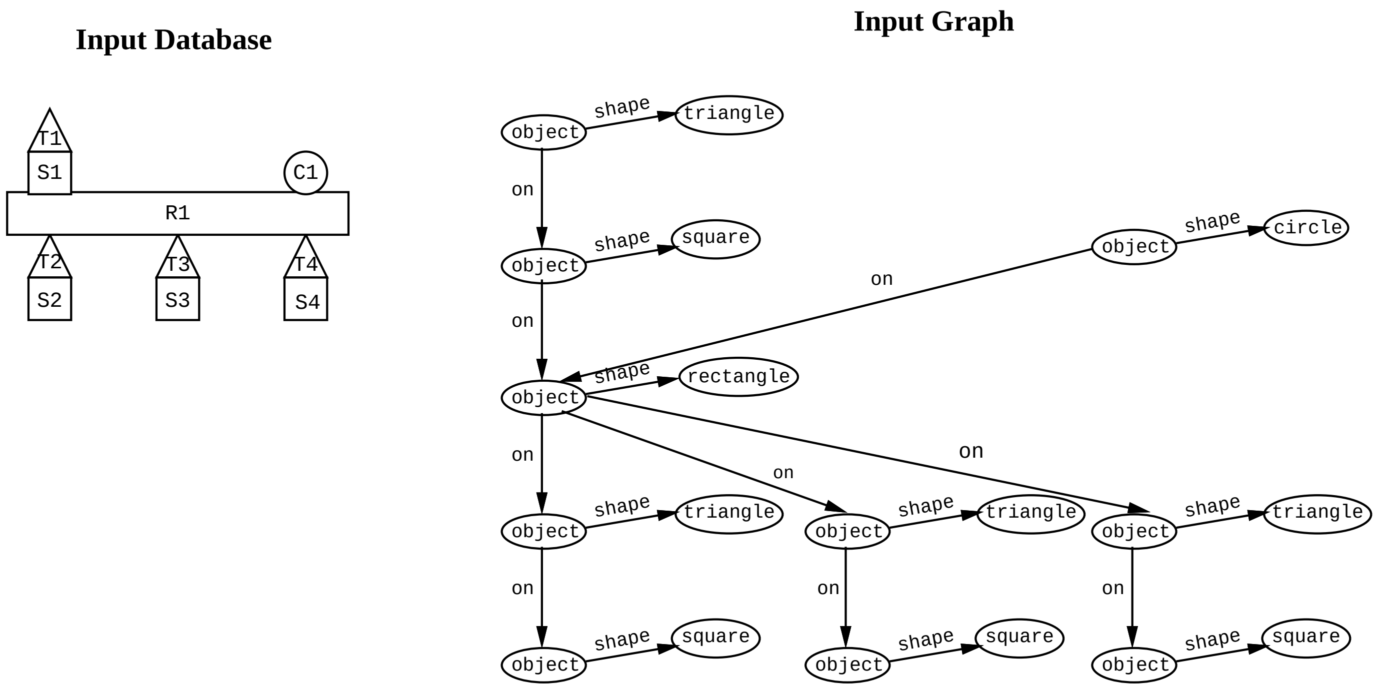


Figure 1: Example substructure in graph form.

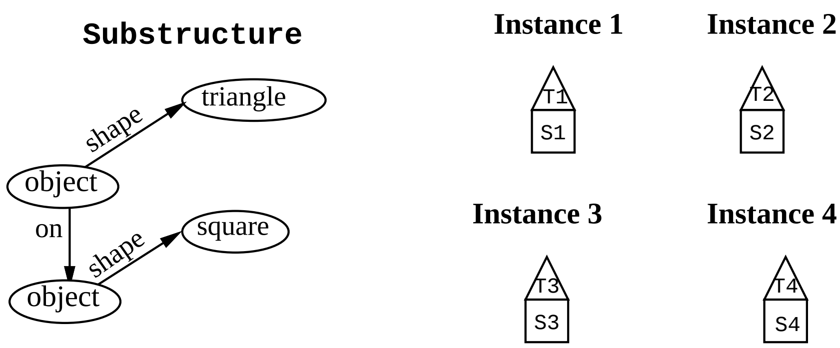


Figure 2: Instances of the substructure.

the other hand, domain-specific knowledge can assist the discovery process by focusing search and can also help make the discovered substructures more meaningful to the user. In this paper we first describe a domain-independent method of performing substructure discovery, and then discuss ways in which the discovery process can be directed with domain-specific information.

In the original version of the SUBDUE system, four domain-independent heuristics were used to evaluate a substructure: cognitive savings, compactness, connectivity and coverage [8]. *Cognitive savings* measures the amount of data compression obtained by describing the original database in terms of the discovered substructure. *Complexity* represents the size or complexity of the discovered substructure. *Compactness* represents the degree to which instances of the substructure are isolated from the rest of the graph, and *Coverage* measures the amount of the original graph that is covered by instances of the discovered substructure. The value of a substructure s for an input graph G can be computed as the weighted product of the four heuristics, as indicated by the formula:

$$value(s) = cognitive\_savings(s)^a \times compactness(s)^b \times connectivity(s)^c \times coverage(s)^d.$$

The exponents are used by a domain expert to weight the individual heuristics.

Although these four heuristics were shown to discover interesting and repetitive substructure in several domains, they may not provide the needed insight for discovery in all possible domains. In addition, the use of these heuristics is largely subsumed by the incorporation of the Minimum Description Length principle. In the current version of our system, the only domain-independent evaluation measure we employ is the Minimum Description Length principle, described in the next section. Domain-dependent knowledge can be added to the process to direct the system toward substructures of particular interest in a given domain.

4 Minimum Description Length Encoding of Graphs

The minimum description length (MDL) principle introduced by Rissanen [19] states that the best theory to describe a set of data is that theory which minimizes the description length of the entire data set. The MDL principle has been used for decision tree induction [16], image processing [13, 14, 10], concept learning from relational data [4], and learning models of non-homogeneous engineering domains [17].

We demonstrate how the minimum description length principle can be used to discover substructures in complex data. In particular, a substructure is evaluated based on how well it can compress the entire dataset using the minimum description length. We define the minimum description length of a graph to be the number of bits necessary to completely describe the graph.

According to the minimum description length principle, the theory that best accounts for a collection of data is the one that minimizes $I(S) + I(G|S)$, where S is the discovered substructure, G is the input graph, $I(S)$ is the number of bits required to encode the discovered substructure, and $I(G|S)$ is the number of bits required to encode the input graph G with respect to S.

The graph connectivity can be represented by an adjacency matrix. Consider a graph that has n vertices, which are numbered $0, 1, \ldots, n-1$. An $n \times n$ adjacency matrix A can be formed with entry $A[i,j]$ set to 0 or 1. If $A[i,j] = 0$, then there is no connection from vertex i to vertex j. If $A[i,j] = 1$, then there is at least one connection from vertex i to vertex j. Undirected edges are recorded in only one entry of the matrix. The adjacency matrix for the graph in Figure 3 is shown below.

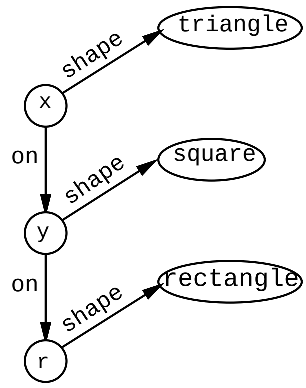


Figure 3: MDL example graph.

$$\begin{array}{r} x \\ triangle \\ y \\ square \\ r \\ rectangle \end{array} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The encoding of the graph consists of the following steps. We assume that the decoder has a table of the l_u unique labels in the original graph G.

1. Determine the number of bits *vbits* needed to encode the vertex labels of the graph. First, we need $(\lg v)$ bits to encode the number of vertices v in the graph. Then, encoding the labels of all v vertices requires $(v \lg l_u)$ bits. We assume the vertices are specified in the same order they appear in the adjacency matrix. The total number of bits to encode the vertex labels is

$$vbits = \lg v + v \lg l_u$$

   For the example in Figure 3, $v = 6$, and we assume that there are $l_u = 8$ unique labels in the original graph. The number of bits needed to encode these vertices is $\lg 6 + 6 \lg 8 = 20.58$ bits.

2. Determine the number of bits *rbits* needed to encode the rows of the adjacency matrix A. Typically, in large graphs, a single vertex has edges to only a small percentage of the vertices in the entire graph. Therefore, a typical row in the adjacency matrix will have much fewer than v 1s, where v is the total number of vertices in the graph. We apply a variant of the coding scheme used by [16] to encode bit strings with length n consisting of k 1s and $(n-k)$ 0s, where $k \ll (n-k)$. In our case, row i $(1 \leq i \leq v)$ can be represented as a bit string of length v containing k_i 1s. If we let $b = \max_i k_i$, then the i^{th} row of the adjacency matrix can be encoded as follows.

   (a) Encoding the value of k_i requires $\lg(b+1)$ bits.

(b) Given that only k_i 1s occur in the row bit string of length v, only $\binom{v}{k_i}$ strings of 0s and 1s are possible. Since all of these strings have equal probability of occurrence, $\lg \binom{v}{k_i}$ bits are needed to encode the positions of 1s in row i. The value of v is known from the vertex encoding.

Finally, we need an additional $\lg(b+1)$ bits to encode the number of bits needed to specify the value of k_i for each row. The total encoding length in bits for the adjacency matrix is

$$\begin{aligned} rbits &= \lg(b+1) + \sum_{i=1}^{v} \lg(b+1) + \lg \binom{v}{k_i} \\ &= (v+1)\lg(b+1) \sum_{i=1}^{v} \lg \binom{v}{k_i} \end{aligned}$$

For the example in Figure 3, $b = 2$, and the number of bits needed to encode the adjacency matrix is $(7 \lg 3) + \lg \binom{6}{2} + \lg \binom{6}{0} + \lg \binom{6}{2} + \lg \binom{6}{0} + \lg \binom{6}{1} + \lg \binom{6}{0} = 21.49$ bits.

3. Determine the number of bits *ebits* needed to encode the edges represented by the entries $A[i,j] = 1$ of the adjacency matrix A. The number of bits needed to encode entry $A[i,j]$ is $(\lg m) + e(i,j)[1 + \lg l_u]$, where $e(i,j)$ is the actual number of edges between vertex i and j in the graph and $m = \max_{i,j} e(i,j)$. The $(\lg m)$ bits are needed to encode the number of edges between vertex i and j, and $[1 + \lg l_u]$ bits are needed per edge to encode the edge label and whether the edge is directed or undirected. In addition to encoding the edges, we need to encode the number of bits $(\lg m)$ needed to specify the number of edges per entry. The total encoding of the edges is

$$\begin{aligned} ebits &= \lg m + \sum_{i=1}^{v} \sum_{j=1}^{v} \lg m + e(i,j)[1 + \lg l_u] \\ &= \lg m + e(1 + \lg l_u) + \sum_{i=1}^{v} \sum_{j=1}^{v} A[i,j] \lg m \\ &= e(1 + \lg l_u) + (K+1) \lg m \end{aligned}$$

where e is the number of edges in the graph, and K is the number of 1s in the adjacency matrix A. For the example in Figure 3, $e = 5$, $K = 5$, $m = 1$, $l_u = 8$, and the number of bits needed to encode the edges is $5(1 + \lg 8) + 6 \lg 1 = 20$.

The total encoding of the graph takes $(vbits + rbits + ebits)$ bits. For the example in Figure 3, this value is 62.07 bits.

Both the input graph and discovered substructure can be encoded using the above scheme. After a substructure is discovered, each instance of the substructure in the input graph is replaced by a single vertex representing the entire substructure. The discovered substructure is represented in $I(S)$ bits, and the graph after the substructure replacement is represented in $I(G|S)$ bits. SUBDUE searches for the substructure S in graph G minimizing $I(S) + I(G|S)$.

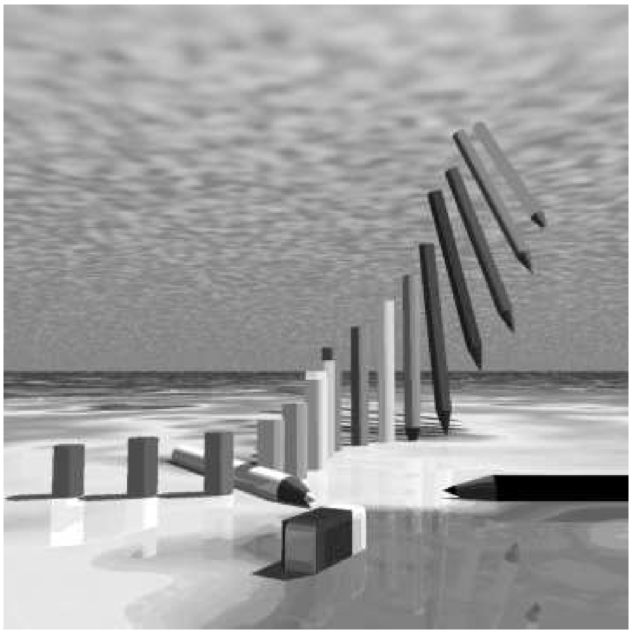


Figure 4: Scene analysis example.

5 Inexact Graph Match

Although exact structure match can be used to find many interesting substructures, many of the most interesting substructures show up in a slightly different form throughout the data. These differences may be due to noise and distortion, or may just illustrate slight differences between instances of the same general class of structures. Consider the image shown in Figure 4. The pencil and the cube would make ideal substructures in the picture, but an exact match algorithm may not consider these as strong substructures, because they rarely occur in the same form and level of detail throughout the picture.

Given an input graph and a set of defined substructures, we want to find those subgraphs of the input graph that most closely resemble the given substructures, and we want to associate a similarity measure with a pair of graphs consisting of a given substructure and a subgraph of the input graph. We adopt the approach to inexact graph match given by Bunke and Allermann [1].

In this inexact match approach, each distortion of a graph is assigned a cost. A distortion is described in terms of basic transformations such as deletion, insertion, and substitution of vertices and edges. The cost for deleting (or inserting) a vertex with label A is denoted by $\textsc{delvertex}(A)$ (or $\textsc{insvertex}(A)$). The cost for substituting a vertex with label A by a vertex with label B is given as $\textsc{subvertex}(A, B)$. Similarly, $\textsc{subedge}(a, b)$ is the cost for an edge substitution. Defining costs for edge deletion and insertion is slightly more difficult since these transformations are dependent on the bordering vertices. When a vertex is deleted, then all of its incoming and outgoing edges vanish also. Conversely, when inserting a vertex, there are usually edges to be inserted to properly embed this vertex in the rest of the graph. As a consequence, we define $\textsc{deledge}(a)$ and $\textsc{insedge}(a)$ as the costs for deletion and insertion of an edge with label a while none of the bordering vertices are deleted or inserted, respectively. In addition, $\textsc{deledge}'(a)$ and $\textsc{insedge}'(a)$ are corresponding costs for edge deletion and insertion when at least one of the bordering vertices has been deleted or

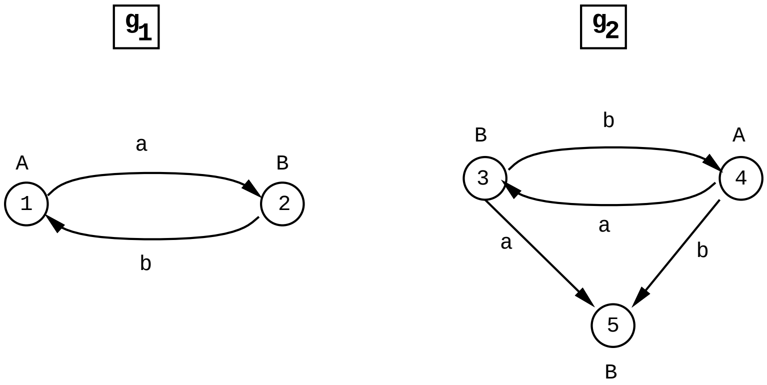


Figure 5: Two similar graphs g_1 and g_2.

inserted, respectively. Specific substitution costs provide a means for the user to express a priori knowledge about the problem domain. Low costs should indicate high likelihood for a particular transformation and vice versa.

An inexact graph match between two graphs g_1 and g_2 maps g_1 to g_2 such that g_2 is interpreted as a distorted version of g_1. Formally, an inexact graph match is a one-to-one mapping $f : N_1 \rightarrow N_2 \cup \{\lambda\}$, where N_1 and N_2 are the sets of vertices of g_1 and g_2, respectively. A vertex $v \in N_1$ that is mapped to λ (i.e., $f(v) = \lambda$) is deleted. That is, it has no corresponding vertex in g_2. Given a set of particular distortion costs as discussed above, we define the cost of an inexact graph match $cost(f)$, as the sum of the cost of the individual transformations resulting from f, and we define $matchcost(g_1, g_2)$ as the value of the least-cost function that maps graph g_1 onto graph g_2.

Given g_1, g_2, and a set of distortion costs, the actual computation of $matchcost(g_1, g_2)$ can be performed using a search procedure. A state in the search tree corresponds to a partial match that maps a subset of the vertices of g_1 to a subset of the vertices in g_2. Initially, we start with an empty mapping at the root of the search tree. Expanding a state corresponds to adding a pair of vertices, one from g_1 and one from g_2, to the partial mapping constructed so far. A final state in the search tree is a match that maps all vertices of g_1 to g_2 or to λ. The complete search tree for the match between the example graphs in Figure 5 is shown in Figure 6. For this example we assign a value of 1 to each distortion cost. The numbers in circles in this figure represent the cost of a state. As we are eventually interested in the mapping with minimum cost, each state in the search tree gets assigned the cost of the partial mapping that it represents. Thus the goal state to be found by our tree search procedure is the final state with minimum cost among all final states. From Figure 6 we conclude that the minimum cost inexact graph match of g_1 and g_2 is given by the mapping $f(1) = 4$, $f(2) = 3$. The cost of this mapping is 3.

Given graphs g_1 with n vertices and g_2 with m vertices, $m \geq n$, the complexity of the full inexact graph match is $O(n^{m+1})$. Because this routine is used heavily throughout the discovery and evaluation process, the complexity of the algorithm can significantly degrade the performance of the system.

To improve the performance of the inexact graph match algorithm, we extend Bunke's approach by applying a branch-and-bound search to the tree. The cost from the root of the tree to a given node in the search tree is computed as described above. Vertices are considered for pairings in order from the most heavily connected vertex to the least connected, as this constrains the remaining

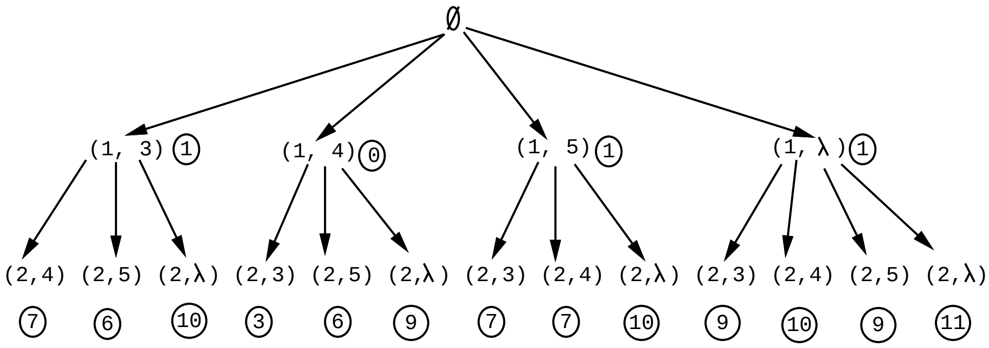


Figure 6: Search tree for computing matchcost(g_1, g_2) from Figure 4.

match. Because branch-and-bound search guarantees that the first solution found is the least-cost solution [18], the search ends as soon as the first complete mapping is found. In the best case, the complexity of the improved inexact graph match is linear in the number of vertices in the larger graph.

In addition, the user can place a limit on the number of search nodes considered by the branch-and-bound procedure (defined as a function of the size of the input graphs). Once the number of nodes expanded in the search tree reaches the defined limit, the search resorts to hill climbing using the cost of the mapping so far as the measure for choosing the best partial mapping at a given level. By defining such a limit, significant speedup can be realized at the expense of accuracy for the computed match cost.

6 Experiments

The experiments in this section evaluate SUBDUE's ability to perform discovery using the MDL principle. Examples are drawn from several domains including scene analysis, chemical analysis and CAD circuit analysis, and analysis of artificially-generated structural databases.

6.1 Domains

6.1.1 Chemical Compound Analysis

Chemical compounds are rich in structure. Identification of the common and interesting substructures can benefit scientists by identifying recurring components, simplying the data description, and focusing on substructures that stand out and merit additional attention.

Chemical compounds are represented graphically by mapping individual atoms, such as carbon and oxygen, to labeled vertices in the graph, and by mapping bonds between the atoms onto labeled edges in the graph. Figures 7, 8, and 9 show the graphs representing the chemical compound databases for cortisone, rubber, and a portion of a DNA molecule.

6.1.2 Scene Analysis

Images and scene descriptions provide a rich source of structure. Images that humans encounter, both natural and synthesized, have many structured subcomponents that draw our attention and that help us to interpret the data or the scene.

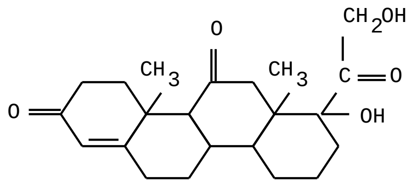


Figure 7: Cortisone.

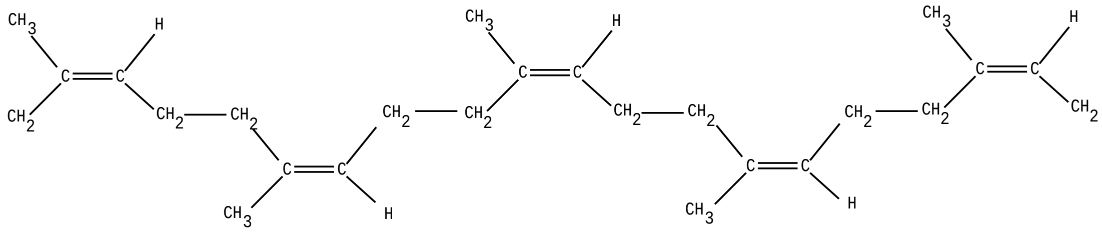


Figure 8: Natural rubber (all-cis polyisoprene).

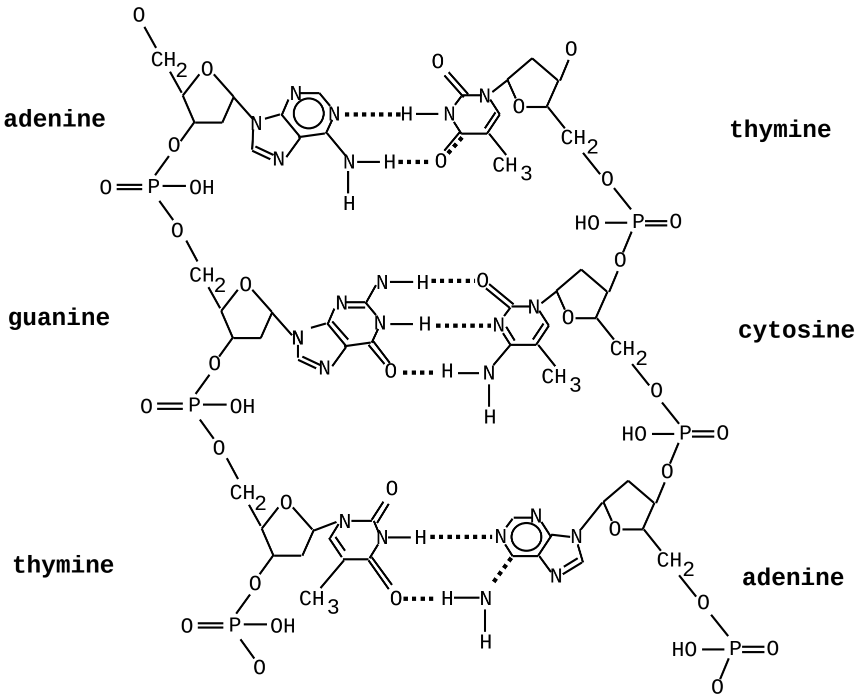


Figure 9: Portion of a DNA molecule.

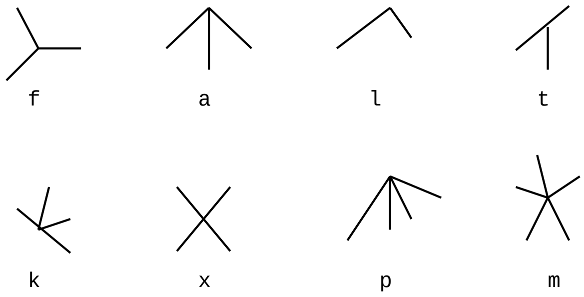


Figure 10: Possible vertices and labels.

Discovering common structures in scenes can be useful to a computer vision system. First, automatic substructure discovery can help a system interpret an image. Instead of working from low-level vertices and edges, SUBDUE can provide more abstract structured components, resulting in a hierarchical view of the image that the machine can analyze at many levels of detail and focus, depending on the goal of the analysis. Second, substructure discovery that makes use of an inexact graph match can help identify objects in a 2D image of a 3D scene where noise and orientation differences are likely to exist. If an object appears often in the scene, the inexact graph match driving the SUBDUE system may capture slightly different views of the same object. Although an object may be difficult to identify from just one 2D picture, SUBDUE will match instances of similar objects, and the differences between these instances can provide additional information for identification. Third, substructure discovery can be used to compress the image. Replacing common interesting substructures by a single vertex simplifies the image description and reduces the amount of storage necessary to represent the image.

To apply SUBDUE to image data, we extract edge information from the image and construct a graph representing the scene. The graph representation consists of eight types of vertices and two types of arcs (*edge* and *space*). The vertex labels (f, a, l, t, k, x, p, and m) follow the Waltz labelings [23] of junctions of edges in the image and represent the types of vertices shown in Figure 10. An *edge* arc represents the edge of an object in the image, and a *space* arc links non-connecting objects together. The *edge* arcs represent an edge in the scene that connects two vertices, and the *space* arcs connect the closest vertices from two disjoint neighboring objects. Distance, curve, and angle information has not been included in the graph representation, but can be added to give additional information about the scene. Figure 11 shows the graph representation of a portion of the scene depicted in Figure 4. In this figure, the *edge* arcs are solid and the *space* arcs are dashed.

6.1.3 CAD Circuit Analysis

In this domain, we employ SUBDUE to find circuit components in CAD circuit data. Discovery of substructures in circuit data can be a valuable tool to an engineer who is attempting to identify common reusable parts in a circuit layout. Replacing individual components in the circuit description by larger substructure descriptions will also simplify the representation of the circuit.

The data for the circuit domain was obtained from National Semiconductor, and consists of a set of components making up a circuit as output by the Cadence Design System. The particular circuit used for this experiment is a portion of an analog-to-digital converter.

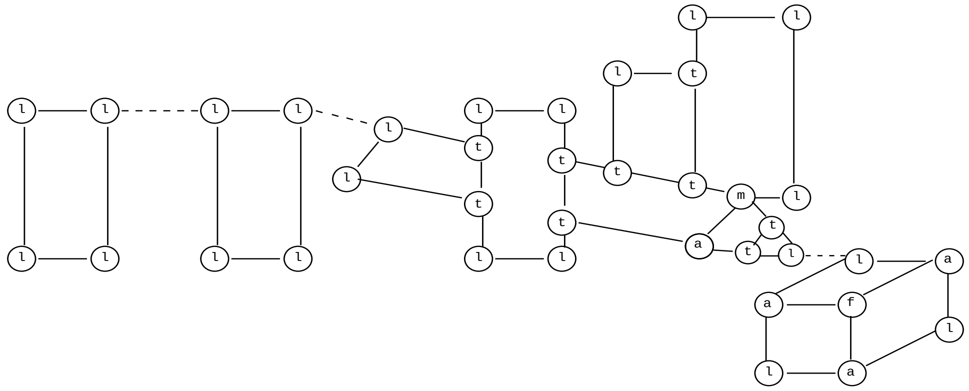


Figure 11: Portion of graph representing scene in Figure 4.

6.1.4 Artificial Domain

In the final domain, we artificially generate graphs to evaluate SUBDUE's ability to discover substructures capable of compressing the graph. First, four substructures are created with random vertex and edge labels, ranging in size from four nodes and three edges to seven nodes and nine edges. Next, these substructures are embedded in larger graphs whose size is 15 times the size of the substructure. The graphs vary across four parameters: number of possible vertex and edge labels (one times and two times the number of labels used in the substructure), connectivity of the substructure (1 or 2 external connections), coverage of the instances (60% and 80%), and the amount of distortion in the instances (0, 1 or 2 distortions). This yields a total of 96 graphs (24 for each different substructure).

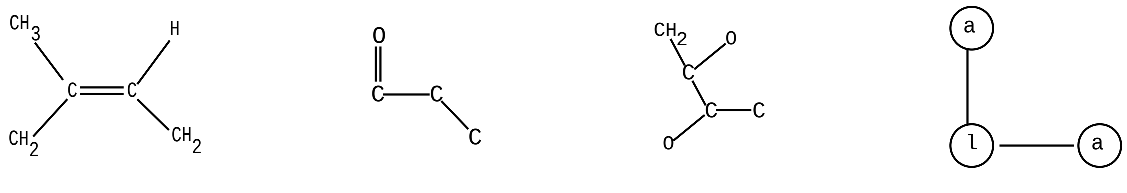
6.2 Experimental Results

In these experiments, we test SUBDUE's ability to compress a structural database. Using a beam width of 4 and SUBDUE's pruning mechanism, we applied the discovery algorithm to each of the databases mentioned above. We repeat the experiment with match thresholds ranging from 0.0 to 1.0 in increments of 0.1. Table 1 lists the description length (DL) of the original graph, the description length of the best substructure discovered by SUBDUE, and the value of compression. Compression here is defined as $\frac{\text{DL of compressed graph}}{\text{DL of original graph}}$. Figure 12 shows the actual discovered substructures for the first four datasets.

As can be seen from Table 1, SUBDUE was able to reduce the database to slightly larger than $\frac{1}{4}$ of its original size in the best case. The average compression value over all of these domains (treating the artificial graphs as one value) is 0.62. The results of this experiment demonstrate that the substructure discovered by SUBDUE can significantly reduce the amount of data needed to represent an input graph. In addition, this compression indicates by the MDL principle that the discovered substructures are the most useful for concisely describing the original database. We expect that compressing the graph using combinations of substructures and hierarchies of substructures will realize even greater compression in some databases.

Database	$DL_{original}$	$Threshold_{optimal}$	$DL_{compressed}$	Compression
Rubber	371.78	0.1	95.20	0.26
Cortisone	355.03	0.3	173.25	0.49
DNA	2427.93	1.0	2211.87	0.91
Pencils	1592.33	1.0	769.18	0.48
CAD – M1	4095.73	0.7	2148.8	0.52
CAD – S1SegDec	1860.14	0.7	1149.29	0.62
CAD – S1DrvBlk	12715.12	0.7	9070.21	0.71
CAD – BlankSub	8606.69	0.7	6204.74	0.72
CAD – And2	427.73	0.1	324.52	0.76
Artificial (avg. over 96 graphs)	1636.25	0.0...1.0	1164.02	0.71

Table 1: Graph compression results.



(a) (b) (c) (d)

Figure 12: Best substructure for (a) rubber database, (b) cortisone datab ase, (c) DNA database, and (d) image database.

7 Hierarchical Concept Discovery

After a substructure is discovered, each instance of the substructure in the input graph can be replaced by a single vertex representing the entire substructure. The discovery procedure can then be repeated on the compressed data set, resulting in new interesting substructures. If the newly discovered substructures are defined in terms of existing substructure concepts, the substructure definitions form a hierarchy of substructure concepts.

Once SUBDUE selects a substructure, all vertices that comprise the exact instances of the substructure are replaced in the graph by a single vertex representing the discovered substructure. Edges connecting vertices outside the instance to vertices inside the instance now connect to the new vertex (information about the endpoint of each edge inside the substructure must be maintained). Edges internal to the instance are removed. If there exists overlap between substructure instances, edges must be added that connect the overlapping instances and provide specific information about the overlap. The discovery process is then applied to the compressed data. If a hierarchical description of concepts is particularly desired, heavier weight can be given to substructures which utilize previously discovered substructures. The increased weight reflects increased attention to this substructure.

To demonstrate the ability of SUBDUE to find a hierarchy of substructures, we let the system make three passes through all of the described databases. Figure 9 shows a portion of a DNA molecule from the chemical compound analysis domain. This picture represents two chains of a double helix, using three pairs of bases which are held together by hydrogen bonds. Figure 13 shows the substructures found by SUBDUE after each of three passes through the data. Note that, on the third pass, SUBDUE linked together the instances of the substructure in the second pass to find the chains of the double helix.

While replacing portions of the input graph with the discovered substructures compresses the data and provides a basis for discovering hierarchical concepts in the data, the substructure replacement procedure becomes more complicated when fuzzy concepts are discovered. When inexact instances of a discovered concept are replaced by a single vertex in the database, all distortions of the graph (differences between the instance graph and the substructure definition) must be attached as annotations to the vertex label. This adds to the complexity of the graph match routine applied in later passes through the database.

8 Domain Knowledge

Since SUBDUE is likely to find substructures which are irrelevant to the domain, domain knowledge can be added to guide the discovery process and to separate the important substructures from the irrelevant. The compression using domain knowledge can increase the chance of finding interesting substructures and realize greater compression.

The domain knowledge is represented using a hierarchy. The nodes of the hierarchy can be classified as either primitive (nondecomposable) or nonprimitive. The primitive nodes reside in the lowest level, i.e., the leaves, and all nonprimitive nodes reside in the higher levels of hierarchy. The primitive nodes represent basic elements of the domain knowledge, whereas the nonprimitive nodes represent substructures which consist of a conglomeration of primitive nodes and/or lower-level nonprimitive nodes. The branches of the hierarchy represent the hierarchical relationship between substructures and their components. The hierarchy for a particular domain is supplied by a domain expert. The substructures in the hierarchy and their functionalities are well known in the context of that domain.

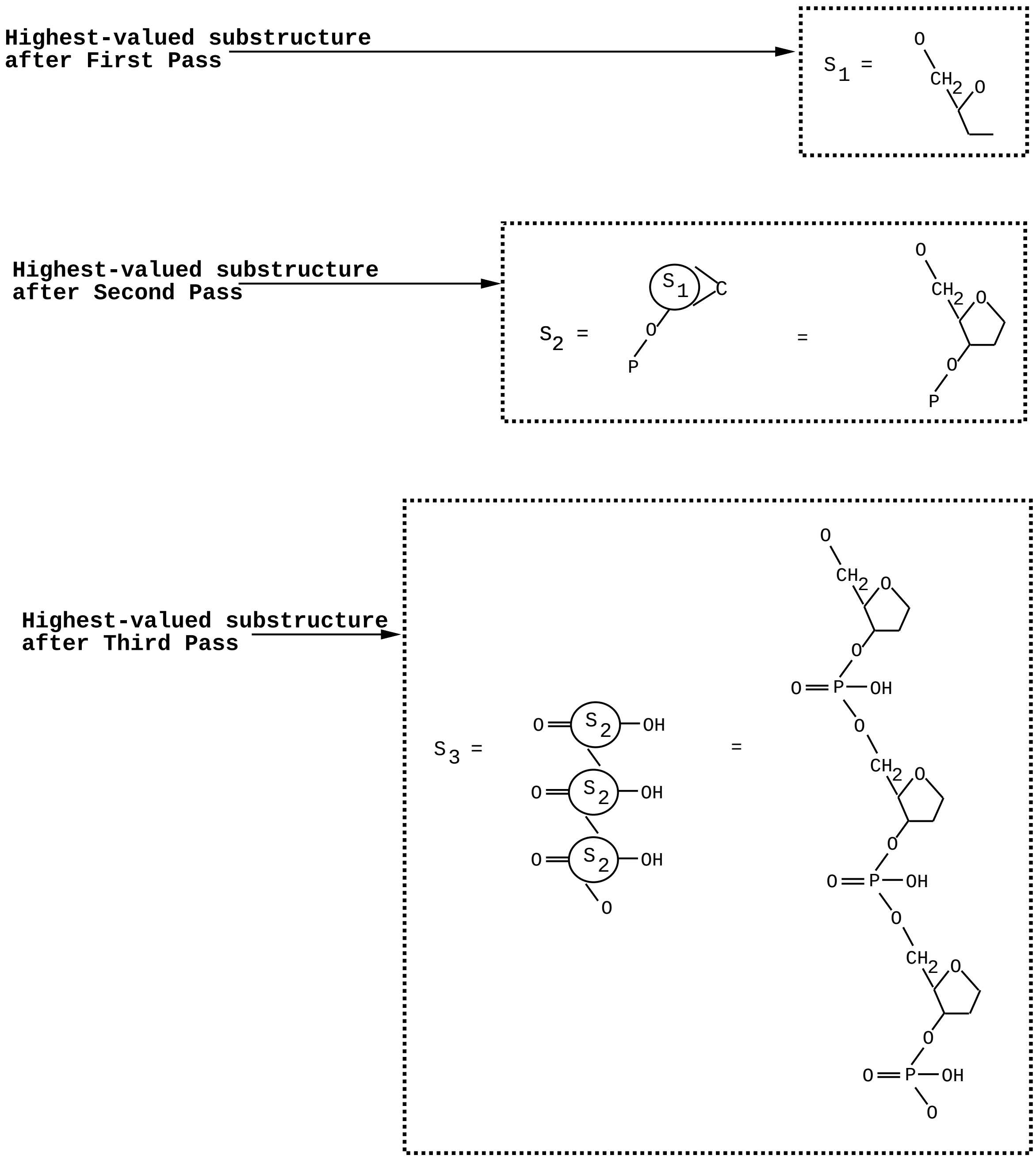


Figure 13: Hierarchical discovery in DNA data.

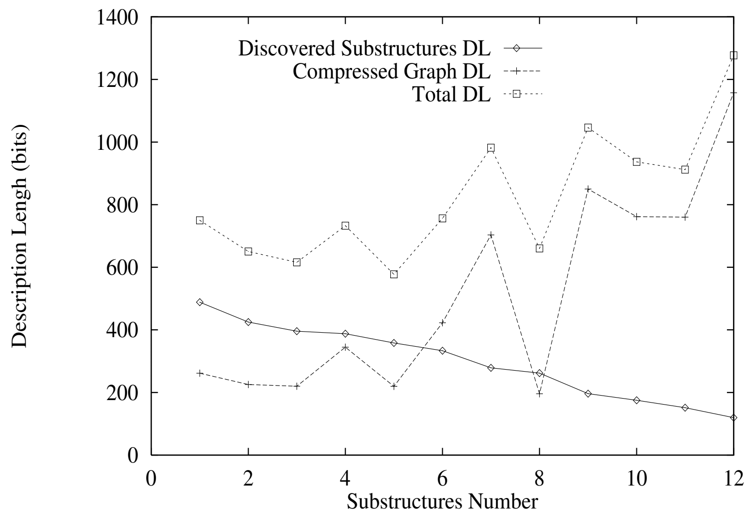


Figure 14: Description length (DL) of discovered substructures and the corresponding compressed graph using domain knowledge in the CAD domain.

In this approach, SUBDUE performs the discovery process using both the heuristics, which is domain independent, and the domain knowledge. Using the domain knowledge, the process begins by matching the input graph's nodes to the leaves of the hierarchy, and if the leaves do not match with the input nodes, the higher level nodes of the hierarchy are pursued. SUBDUE compares the discovered substructure with the substructures matched in the domain knowledge and selects the substructure which gives the best match and most compression.

After each iteration of the discovery process, the SUBDUE system discovers at most two best substructures, one based on the heuristics and the other based on the domain knowledge. These substructures are used to compress the input graph. To avoid obtaining a heuristic-based substructure of little use, the heuristic-based substructure will not be used if its amount of local compression (as computed with respect to the previous iteration) is less than a predefined domain threshold. Either one or two compressed graphs can be obtained in each iteration depending on how many substructures can be used. SUBDUE selects the compressed graph which gives the maximum amount of local compression as the input graph to the next iteration of the discovery process. The compressed graph which has not been selected is put in the list of unprocessed graphs. If after further iterations, SUBDUE obtains a compressed graph whose amount of local compression is smaller than any compressed graph in the list of unprocessed graphs, this compressed graph is put on the list of unprocessed graphs. SUBDUE resumes the discovery process using the compressed graph from the list of unprocessed graphs which has the maximum amount of local compression. This process is repeated until the list of unprocessed graphs is exhausted. The MDL principle is used as a compression measure for both the heuristic-based and the domain knowledge based discovery. The encoding scheme used in the domain knowledge discovery is the same as the scheme used in the heuristic-based discovery.

Figure 14 shows the initial results of using a simple domain knowledge hierarchy for the CAD domain. The figure shows the description length of the discovered substructures, the compressed graph, and the sum of these two, versus the substructures obtained during each iteration of the discovery process. The substructures number represents the substructures used to compress the input graph at some iteration of the discovery process. The substructures numbers increase with decreasing description length of the substructures used for an iteration.

The minimum of the total description length occurred at the substructures number 5, which considers the trade-off between the description length of discovered substructures and the power of those substructures to compress the graph. For substructures number 1, the description length of the discovered substructures is the highest. They compress the graph better and therefore decrease the description length of the compressed graph; however, the total description length is not the lowest, indicating there is some redundancy contained in the discovered substructures. On the other hand, for the substructures number 12, description length of the discovered substructures is the lowest, indicating the discovered substructures are too trivial. The resulting compression is low as indicated by the larger description length of the compressed graph.

Of the four best substructures numbers (5, 3, 2, 8) in terms of lower total description length, numbers 3 and 8 contain a mixture of substructures from the heuristic-based and domain knowledge based discovery. Numbers 5 and 2 contain substructures from only the heuristic-based discovery. Therefore, the domain knowledge is useful in identifying relevant substructures, and SUBDUE's heuristic-based discovery process is capable of finding relevant substructures in terms of their compression performance with respect to the domain knowledge substructures.

9 Conclusions

Extracting knowledge from structural databases requires the identification of repetitive substructures in the data. Substructure discovery provides the toolw necessary to perform this task. The substructures represent new concepts found in the data and a means of reducing the complexity of the representation by abstracting over instances of the substructure. We have shown how the minimum description length (MDL) principle can be used to perform substructure discovery in a variety domains. The use of an inexact graph match allows deviation in the instances of a substructure, and multiple passes through the database can be used to build a hierarchical description of the substructures. Domain-dependent models can be used to guide the discovery process to substructures which are of particular interest in a given domain.

Many of the experiments we present in this paper demonstrate SUBDUE's ability to re-discover concepts that are already known to be significant in a given domain. Future experiments with SUBDUE will allow the system to run on a new database such as NASA's satellite image databases, and the results will be evaluated by an expert familiar with the given data. We expect SUBDUE to generate both significant and surprising results in this type of experiment.

Future work will also take advantage of parallel hardware to speed up the discovery algorithm. Parallelization on a MIMD machine by distributing the search space will allow SUBDUE to scale up to much larger databases without significant increase in processing time.

Acknowledgements

The authors would like to thank Horst Bunke for reviewing this work and providing insight into the inexact graph match algorithm. We would also like to thank Mike Shay at National Semiconductor for providing the circuit data.

References

[1] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

[2] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A bayesian classification system. In *Proceedings of the Fifth International Workshop on Machine Learning*, pages 54–64, 1988.

[3] D. Conklin, S. Fortier, J. Glasgow, and F. Allen. Discovery of spatial concepts in crystallographic databases. In *Proceedings of the ML92 Workshop on Machine Discovery*, pages 111–116, 1992.

[4] M. Derthick. A minimal encoding approach to feature discovery. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 565–571, 1991.

[5] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[6] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[7] L. Holder, D. J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[8] L. B. Holder, D. J. Cook, and H. Bunke. Fuzzy substructure discovery. In *Proceedings of the Ninth International Machine Learning Conference*, pages 218–223, 1992.

[9] E. Jeltsch and H. J. Kreowski. Grammatical inference based on hyperedge replacement. In *Fourth International Workshop on Graph Grammars and Their Application to Computer Science*, pages 461–474, 1991.

[10] Y. G. Leclerc. Constructing simple stable descriptions for image partitioning. *International journal of Computer Vision*, 3(1):73–102, 1989.

[11] R. Levinson. A self-organizing retrieval system for graphs. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 203–206, 1984.

[12] L. Miclet. *Structural Methods in Pattern Recognition*. Chapman and Hall, 1986.

[13] E. P. D. Pednault. Some experiments in applying inductive inference principles to surface reconstruction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1603–1609, 1989.

[14] A. Pentland. Part segmentation for object recognition. *Neural Computation*, 1:82–91, 1989.

[15] G. Piatetsky-Shapiro. *Knowledge Discovery in Databases*. AAAI Press, 1991.

[16] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[17] R. B. Rao and S. C. Lu. Learning engineering models with the minimum description length principle. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 717–722, 1992.

[18] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, 1991.

[19] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Company, 1989.

[20] R. J. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley & Sons, 1992.

[21] J. Segen. Graph clustering and model learning by data compression. In *Proceedings of the Seventh International Machine Learning Workshop*, pages 93–101, 1990.

[22] K. Thompson and P. Langley. Concept formation in structured domains. In D. H. Fisher and M. Pazzani, editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, chapter 5. Morgan Kaufmann Publishers, Inc., 1991.

[23] D. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[24] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 157–210. McGraw-Hill, 1975.

[25] K. Yoshida, H. Motoda, and N. Indurkhya. Unifying learning methods by colored digraphs. In *Proceedings of the Learning and Knowledge Acquisition Workshop at IJCAI-93*, 1993.