Pattern Recognition and String Matching
D. Chen and X. Cheng (Eds.) pp. – - –

Structural Pattern Recognition in Graphs

Lawrence Holder
Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX 76019
E-mail: holder@cse.uta.edu

Diane Cook
Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX 76019
E-mail: holder@cse.uta.edu

Jesus Gonzalez
Instituto Nacional de Astrofisica Optica y Electronica Ciencias Computacionales
Luis Enrique Erro 1, Sta. Maria Tonantzintla, APDO. Postal 51 y 216, C.P. 72840, Puebla, Puebla. Mexico
E-mail: jagonzalez@inaoep.mx

Istvan Jonyer
Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX 76019
E-mail: jonyer@cse.uta.edu

Contents

1 Introduction

Most pattern recognition approaches look for patterns in data represented as independent entities described by attributes. However, the relationships between entities are as important, if not more important, to the recognition of accurate and meaningful patterns. In this chapter we describe an approach to discovering patterns in relational data represented as a graph. Our approach is based on the minimum description length (MDL) principle [28], which measures how well various patterns compress the original database. This approach is implemented in the SUBDUE system. We begin with a discussion of related work. We then describe graph-based discovery, the main discovery algorithm, and the polynomially-constrained inexact graph matching algorithm at the heart of the discovery process. Next, we describe how this technique can also be used for clustering and concept learning. We

illustrate the utility of the approach by applying the clustering and concept learning techniques to DNA and WWW data.

2 Related Work

Systems that deal with structural databases typically do not scale for large databases due to the increased combinatorics inherent in the richer representations. Inductive Logic Programming (ILP) systems can handle structural representations [15], but do not yet scale to large databases. Although ILP systems typically perform supervised learning, some systems (e.g., CLAUDIEN [27] and PROGOL [25]) attempt unsupervised discovery from structural data using techniques common in ILP systems. Our graph-based approach is constrained to run in polynomial time while still discovering relevant patterns [9, 11, 12, 10].

Other approaches to discovery in structural databases have been proposed [7, 20, 29, 33, 35]. Many of these approaches use a knowledge base of concepts to classify the structural data. These systems perform concept learning over examples and categorization of new data, and are typically not designed to perform unsupervised discovery. While the above methods process individual objects one at a time, our method is designed to process the entire structural database, consisting of many objects. Unlike many of the existing methods, SUBDUE can also discover knowledge in a supervised or unsupervised fashion.

The graph-based discovery approach relies on the ability to perform an inexact, or error-tolerant, graph isomorphism between two graphs. Our inexact graph match is based on a branch-and-bound search to find a minimal edit distance [4]. Such approaches have been refined [26, 23, 22], especially in their application to computer vision for matching artifacts of an image represented in graph form. Related work on computing an error-tolerant subgraph isomorphism for a set of graphs optimizes the graph algorithm by first matching common subgraphs of the set of graphs to be matched [24]. We also are interested in finding common subgraphs within a set of graphs, but also in a single graph, and our approach searches for such subgraphs motivated by the desire to find subgraphs that not only optimally compressing the graph, but also represent knowledge of interest to the user.

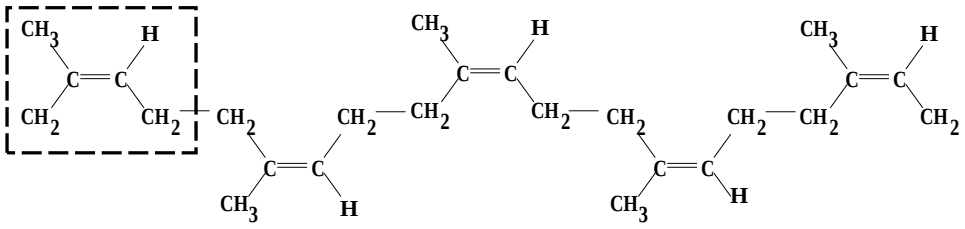

Figure 1: Natural rubber atomic structure.

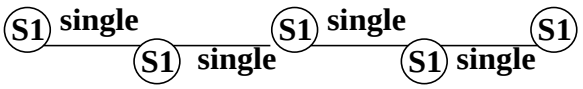

Figure 2: Rubber graph compressed using the discovered substructure.

3 Graph-Based Discovery

The graph-based discovery method discovers structural patterns, or substructures, in a graph representation of data guided by the MDL principle. Based on the MDL principle, the method discovers substructures that compress the original data and represent structural concepts in the data. Once a substructure is discovered, the substructure is used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered substructure. The discovered substructures allow abstraction over detailed structures in the original data. Iteration of the substructure discovery and replacement process constructs a hierarchical description of the structural data.

3.1 Graph Representation

The substructure discovery approach represents structural data as a labeled 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 algorithm. An *instance* of a substructure in a graph is a set of vertices and edges from the input graph that match, graph theoretically, to the graphical representation of the substructure.

Figure 1 shows a sample database that is input to SUBDUE, representing

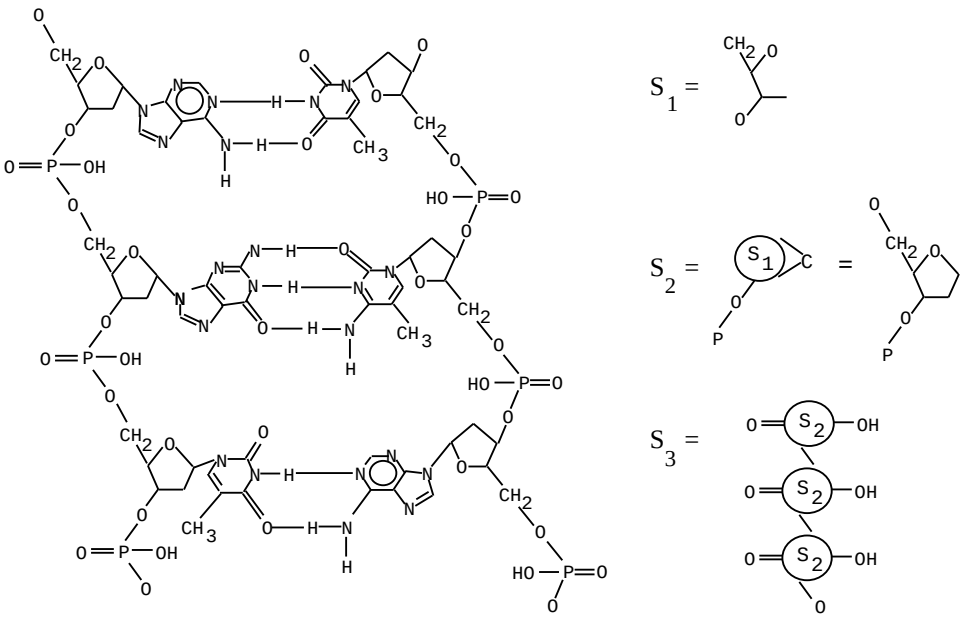

Figure 3: Sample results of Subdue on a partial DNA sequence.

the atomic structure of natural rubber. The input graph represents atoms as vertices, and single or double bonds as labeled undirected edges between the vertices. The highlighted substructure, which we label substructure S1, is selected as the best subgraph to describe the input database. The five instances of the substructure are each replaced by a single vertex representing the discovered concept. The graph that results from compressing the rubber database is shown in Figure 2. This discovery and compression process helps the user understand the database and exploit the knowledge content of the data encoded in the graph vertices and edges.

Figure 3 shows a sample input database containing a portion of a DNA sequence. In this case, atoms and small molecules in the sequence are represented with labeled vertices in the graph, and the single and double bonds between atoms are represented with labeled edges in the graph. SUBDUE discovers substructure S_1 from the input database. After compressing the original database using S_1, SUBDUE discovers substructure S_2, which when used to compress the database further allows SUBDUE to discover substructure S_3. Such repeated application of SUBDUE generates a hierarchical description of the structures in the database.

```
Subdue (Graph, Beam, Limit)
    queue Q = {v | v is a vertex in Graph having a unique label}
    bestSub = first substructure in Q
    repeat
        newQ = {}
        for each substructure S ∈ Q
            newSubs = Extend-Substructure (S, Graph)
                      in all possible ways
            Evaluate (newSubs)
            newQ = newQ ∪ newSubs mod Beam
            Limit = Limit - 1
        if best substructure in newQ better than bestSub
        then bestSub = best substructure in Q
        Q = newQ
    until Q is empty or Limit ≤ 0
    return bestSub
```

Figure 4: SUBDUE's discovery algorithm.

3.2 Discovery Algorithm

SUBDUE uses a variant of beam search [36, 16]. Beam search maintains a limited-length (i.e., *beam* length) queue of the best few patterns found so far. While limiting the queue length yields a suboptimal search, not limiting the queue length may result in an exponential-sized queue. The goal of the search is to find the substructure (pattern) that best compresses the description length of the input graph. A substructure in SUBDUE consists of a substructure definition and all its instances in the graph.

SUBDUE's discovery algorithm is shown in Figure 4 and is given the input graph, the beam length, and a limit on the total number of substructures considered by the algorithm. The initial state of the search is the set of substructures representing each uniquely labeled vertex and its instances. The only search operator is the *Extend-Substructure* operator. As its name suggests, *Extend-Substructure* extends the instances of a substructure in all possible ways by a single edge and a vertex, or by a single edge if both vertices are already in the substructure. The minimum description length (MDL) principle is used to evaluate the substructures.

The search progresses by applying the *Extend-Substructure* operator to each substructure in the current search frontier (queue), which is an ordered list of previously discovered substructures. The resulting frontier, however, does not contain all the substructures generated by the *Extend-Substructure* operator. The substructures are stored on a queue and are ordered based on their ability to compress the graph. The length of the queue is limited by the user-defined *Beam* parameter, where *mod Beam* means that only the best *Beam* substructures are kept on the queue after adding new substructures to the queue. The search terminates upon reaching a user-specified *Limit* on the number of substructures extended, or upon exhaustion of the search space. SUBDUE's run time is polynomial in *Beam* and *Limit*.

Once the search terminates and returns the best substructure, the graph can be compressed using this substructure. The compression procedure replaces all instances of the substructure in the input graph by a single vertex, which represents the substructure. Incoming and outgoing edges to and from the replaced substructure will point to, or originate from, the new vertex that represents the substructure. In our implementation, we do not maintain information on how vertices in each instance were connected to the rest of the graph, which results in lossy compression. Since the goal of substructure discovery is interpretation of the database, maintaining information to reverse the compression is of secondary importance.

The SUBDUE algorithm can be invoked again on this compressed graph. This procedure can be repeated a user-specified number of times, and is referred to as an iteration. The maximum number of iterations that can be performed on a graph cannot be predetermined; however, a graph that has been compressed into a single vertex cannot be compressed further.

To allow SUBDUE to discover substructures of particular interest to a scientist in a specific domain, the user can direct the search with expert-supplied background knowledge [12]. Background knowledge can take the form of known substructure models that may potentially appear in the database, or graph match rules to adjust the cost of each graph transformation. Unlike other existing approaches to graph-based discovery [8, 20, 30, 33, 37], SUBDUE is effective at finding interesting and repetitive substructures in any structural database with or without domain-specific guidance.

3.3 Minimum Description Length Principle

SUBDUE's search is guided by the minimum description length (MDL) principle developed by Rissanen [28]. According to the MDL heuristic, the best

substructure is the one that minimizes the description length of the graph when compressed by the substructure. This compression is calculated as

$$Compression = \frac{DL(S) + DL(G \mid S)}{DL(G)}$$

where $DL(G)$ is the description length of the input graph, $DL(S)$ is the description length of the substructure, and $DL(G \mid S)$ is the description length of the input graph compressed by the substructure. The search algorithm attempts to maximize the value of the substructure, which is simply the inverse of the Compression. The description length of a graph is calculated here as the number of bits needed to encode an adjacency matrix representation of the graph. Additional details of the encoding scheme are reported in the literature [9].

3.4 Inexact Graph Match

Because instances of a substructure can appear in different forms throughout the database, an inexact graph match based on [4] is used to identify substructure instances. Performing an inexact match allows the discovered substructures to abstract away minor variations in the substructure instances. Subgraphs of the original database are considered to be instances of a substructure definition if the edit distance between the two graphs, or the cost of transforming the potential instance into a graph that is isomorphic with the substructure definition, does not exceed a user-defined value. Transformations between graphs can include addition or deletion of vertices, addition or deletion of edges, vertex label substitutions, and edge label substitutions. Each transformation is assigned a cost which can be customized by the user for a specific application.

Formally, an inexact graph match from graph g_1 to graph g_2 is a 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, the vertex has no corresponding vertex in g_2 and therefore maps to nothing (λ). Given a set of graph transformation costs, 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. A SUBDUE parameter, the match threshold t, can be specified with a value between 0 and 1. A graph is considered to be an instance of a substructure if the matchcost of the two graphs is no more than t times the size of the

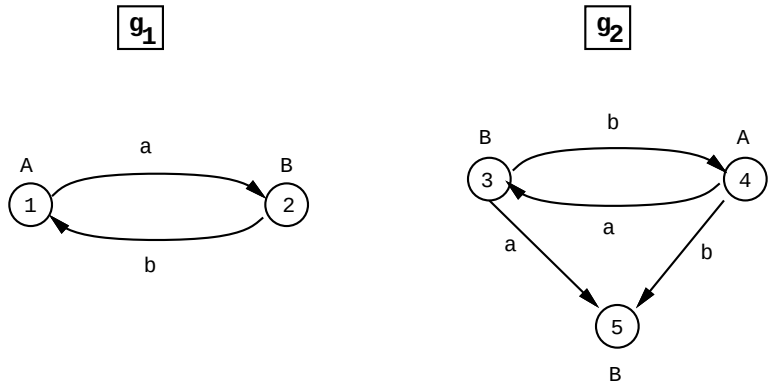

Figure 5: Two similar graphs g_1 and g_2.

larger graph. A threshold value of 0 constrains the algorithm to allow only exact matches, and a threshold value of 1 allows any two graphs to match.

Given g_1, g_2, and a set of distortion costs, the actual computation of $matchcost(g_1, g_2)$ can be determined using a tree 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 $g_2 \cup \{\lambda\}$, 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 of the example 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 is assigned the cost of the corresponding partial mapping. Thus the goal state to be found by our tree search procedure is the full mapping, or leaf node, with the minimum cost among all full mappings. 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 4. 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 throughout the discovery process, the complexity of the algorithm can significantly degrade the performance of the system. We offer a number of enhancements that improve the efficiency of

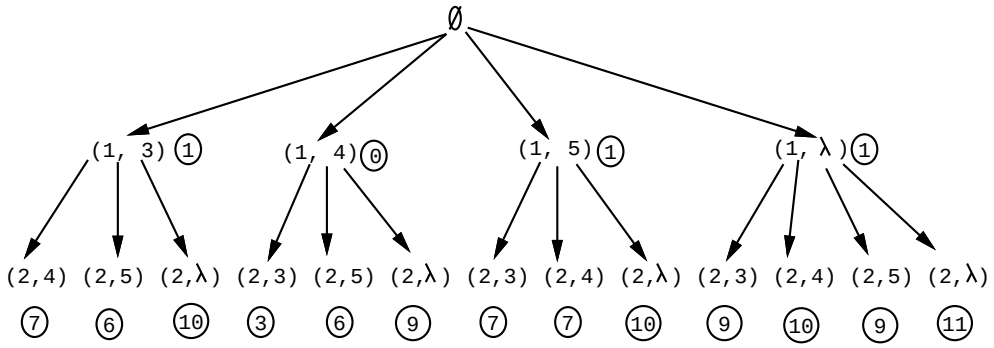

Figure 6: Search tree for computing matchcost(g_1,g_2) from Figure 5.

this algorithm. First, we apply a branch-and-bound search algorithm to the search space in which partial mappings are considered in nondecreasing order by match cost. The cost from the root to a given node is calculated as the cost of all distortions corresponding to the partial mapping for that node. Vertices from the matched graphs are considered in nonincreasing order by degree. Because branch-and-bound GUARANTEEs an optimal solution, the search ends when the first complete mapping (leaf node) is found.

Second, the user can limit the number of search nodes considered by the branch-and-bound procedure. The limit is defined as a function of n, the size of the larger graph. Once the number of nodes expanded in the search tree reaches the defined limit, the search switches to hill climbing or greedy searching using the cost of the partial mapping as the measure for choosing the best node at a given level.

Third, the match threshold itself can be used to prune portions of the search space. Because no graph transformations can be assigned a negative cost, as soon as a node is reached with a match cost greater than the allowed value, the subtree rooted at that node is removed from consideration.

Employing computational constraints such as a bound on the number of substructures considered (s) and the number of partial mappings considered during an inexact graph match (x), SUBDUE is constrained to run in polynomial time. The worst-case run time of the system is the product of the number of generated substructures, the number of instances of each substructure, and the number of partial mappings considered during graph match. This expression is equal to $(\sum_{i=1}^{s} i*((v-1)-(i-1)))*(v(s-1))*x$,

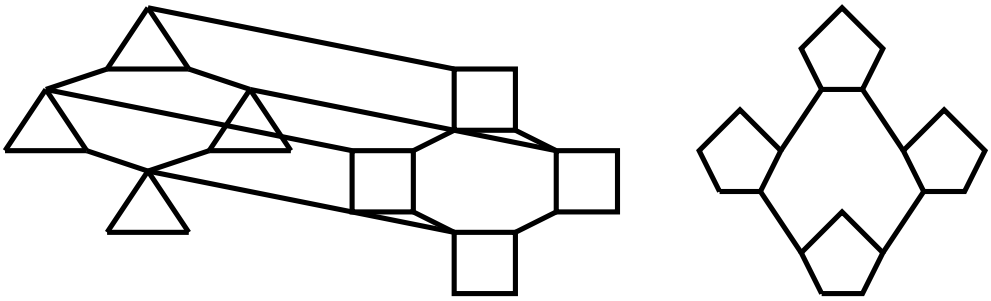

Figure 7: Sample geometric domain for clustering.

where v represents the number of vertices in the input graph.

4 Graph-Based Clustering

Cluster analysis has been studied and developed for model fitting, hypothesis generation and testing, data exploration and reduction, prediction based on groups, and finding true topologies [1]. The purpose of applying clustering to a database is to gain a better understanding of the data, in many cases by highlighting hierarchical topologies. An example of a hierarchical clustering is the classification of vehicles into groups such as cars, trucks, motorcycles, tricycles, and so on, which are then further subdivided into smaller groups based on observed traits. Although a number of relatively successful clustering systems have been constructed [6, 17, 33], few existing systems address the problem of clustering in discrete-valued, structural databases. Our approach centers on discrete-valued, structural databases that are represented as graphs.

One approach to structural clustering is to perform multiple iterations of SUBDUE, constraining the discovery algorithm to not reuse substructures discovered in previous iterations, until the whole graph is exhausted. Each iteration yields one cluster (substructure), which is used to compress the graph. This cluster can be inserted into a *classification lattice*. This lattice structure contrasts with previous research that generates classification trees, and is partially due to the fact that SUBDUE has the unique capability to represent more elaborate relationships in the data using graph structure.

4.1 Example

As an example, SUBDUE can be used to cluster the data shown in Figure 7. Visually, the input data consists of triangles, squares and pentagons, where vertices and edges in the figure map to vertices and edges in the input graph. SUBDUE first discovers the substructure describing the pentagon pattern in the input graph, which is inserted as a child cluster of the root node in the lattice. During the next two iterations, the square shape and triangle shape as discovered and inserted as children of the root. In the fourth iteration, SUBDUE returns the substructure describing two pentagon shapes connected by a single edge. This cluster is inserted into the classification lattice as the child of the cluster describing the pentagon, because the pentagon appears in the cluster definition. There are two links connecting this new cluster to its parent, because the parent cluster definition appears twice. In the last iteration, a substructure is discovered that contains a pair of squares connected by an edge, a pair of triangles connected by an edge, and an edge connecting the two pairs. This cluster is inserted as a child of two clusters from the first level of the lattice. The resulting lattice is depicted in Figure 8.

4.2 Application to DNA

We applied graph-based clustering to the DNA sequence data from Figure 3. The resulting lattice is shown in Figure 9. The lattice property is apparent in Figure 9, where the bottom-left nodes have multiple parents. This lattice describes 71% of the DNA sequence shown in Figure 3. As the lattice shows, smaller, more commonly occurring compounds are found first that compose the first level of the lattice. These account for more than 61% of the DNA sequence data. Subsequently identified clusters are based on these smaller clusters that are either combined with each other, or with other atoms or molecules to form a new cluster. The second level of the lattice extends the conceptual clustering description such that an additional 7% of the DNA is covered.

5 Graph-Based Concept Learning

We have extended our unsupervised discovery methods to perform supervised graph-based relational concept learning. Logic-based systems have dominated the area of relational concept learning, especially Inductive Logic Programming (ILP) systems. However, first-order logic can also be repre-

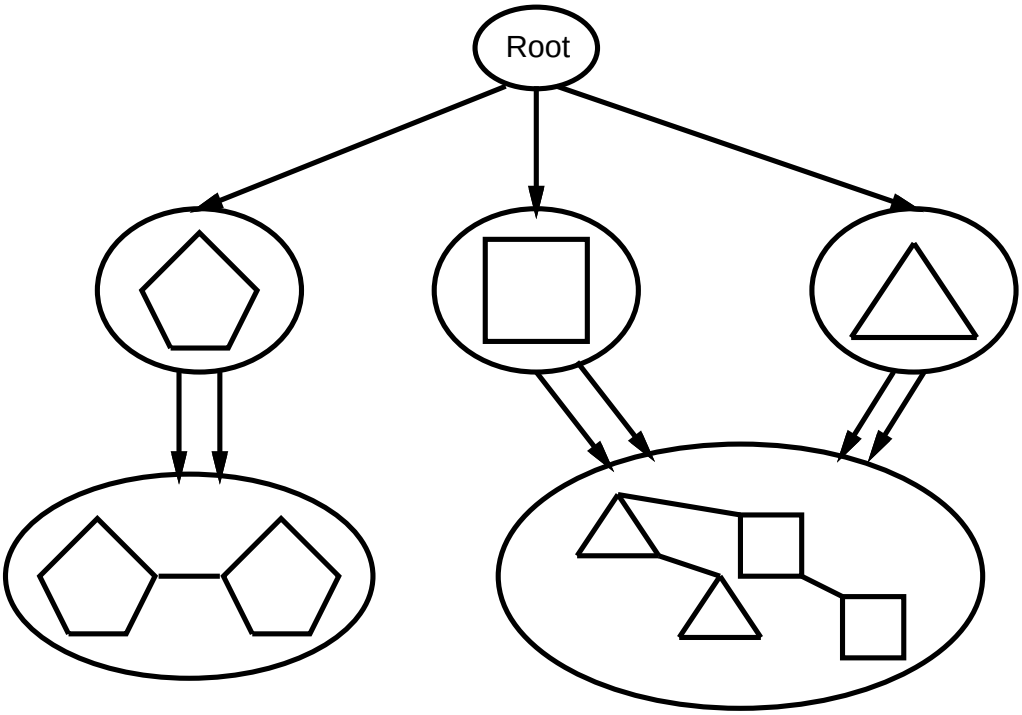

Figure 8: Clustering lattice for geometric domain in Figure 7.

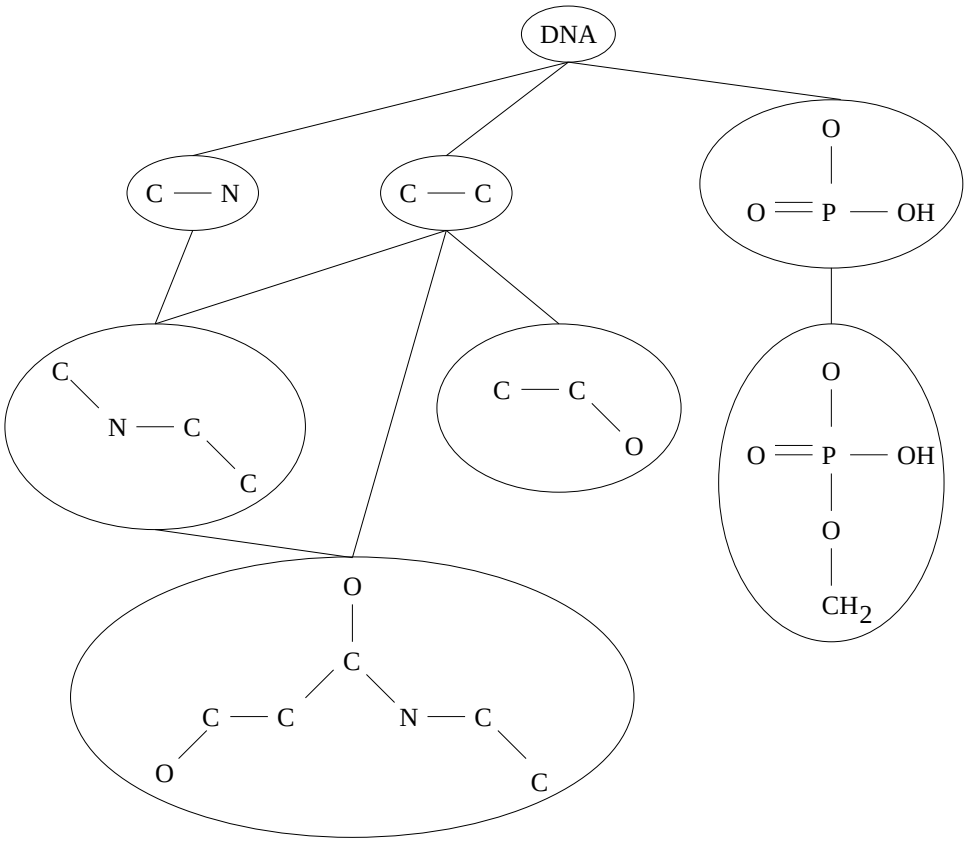

Figure 9: Clustering lattice for DNA domain.

sented as a graph, and in fact, first-order logic is a subset of what can be represented using graphs. Therefore, learning systems using graphical representations have the potential to learn richer concepts if they can handle the increased size of the hypothesis space.

Few general-purpose learning methods use a formal graph representation for knowledge, perhaps because of the arbitrary expressiveness of a graph and the inherent NP-hardness of typical graph-based learning routines. Graph-based learning methods for pattern recognition in chemical domains [2, 32, 34] have been successful due to the natural graphical description of chemical compounds. Some unsupervised learning methods (e.g., SUBDUE [9] and WARMR [14]) use a graph representation, but no domain-independent supervised concept learning systems use a graph representation (although there is some theoretical work on graph-based concept learning [21]).

The main challenge in adding concept-learning capabilities to SUBDUE is the inclusion of "negative" examples into the process. Substructures that describe the positive examples; but not negative examples, are likely to represent the target concept. Therefore, the SUBDUE concept learner (which we will refer to as SUBDUECL) accepts both positive and negative examples in graph format. Since SUBDUECL is an extension to SUBDUE, it uses SUBDUE's core functions to perform graph operations, but the learning process is different. SUBDUECL works as a supervised learner by differentiating positive and negative examples using a set-covering approach instead of graph compression. The hypothesis found by SUBDUECL consists of a set of disjunctions of conjunctions (substructures), i.e., the concept may contain several rules. SUBDUECL forms one of these conjunctions (rules) in each iteration. Positive example graphs that are described by the substructure found in a previous iteration are removed from the graph for subsequent iterations.

SUBDUECL uses an evaluation formula to give a value to all the generated substructures. This formula assigns a value to a substructure according to how well it describes the positive examples (or a subset of the positive examples) without describing the negative examples. Then, positive examples covered by the substructure increase the substructure value while negative examples decrease its value. In this formula the positive examples that are not covered and the negative examples covered by the substructure are considered errors, because the ideal substructure would be one covering all the positive examples without covering any negative example. Then, the substructure value is calculated as follows:

$$value = 1 - Error$$

where the error is calculated with respect to the positive and negative examples covered by the substructure using the following formula:

$$Error = \frac{\#PosEgsNotCovered + \#NegEgsCovered}{\#PosEgs + \#NegEgs}$$

$\#PosEgsNotCovered$ is the number of positive examples not covered by the substructure, and $\#NegEgsCovered$ is the number of negative examples covered by the substructure. $\#PosEgs$ is the number of positive examples remaining in the training set (remember that the positive examples that have already been covered in a previous iteration were removed from the training set), and $\#NegEgs$ is the total number of negative examples. This number does not change, because negative examples are not removed from the training set. Of two substructures with the same error, the substructure covering more positive examples is preferred.

In addition to replacing the compression-based evaluation measure with error-based measure mentioned above, the SUBDUECL algorithm differs from the original SUBDUE algorithm from Figure 4 in that SUBDUECL is called multiple times in a set-covering framework, each time adding a new substructure to the disjunctive hypothesis and removing covered positive examples. This process continues until either all positive examples are covered or no substructure exists discriminating the remaining positive examples from the negative examples (i.e., noise exists in the data).

5.1 Example

As an example of SUBDUECL and its comparison to the ILP systems FOIL [5] and PROGOL [25], we present results from the chess endgame database available at the UCI Machine Learning Repository [3]. The chess domain consists of 20,000 examples of row-column positions for a white king, white rook and black king such that the black king is in check (positive) or not (negative). Therefore, if white's turn is next, then the positive examples are illegal configurations. The relational information in this domain consists of adjacency relations between board positions and less-than or equal relations between row and column numbers (0-7). Both FOIL and PROGOL extensionally define the three relations.

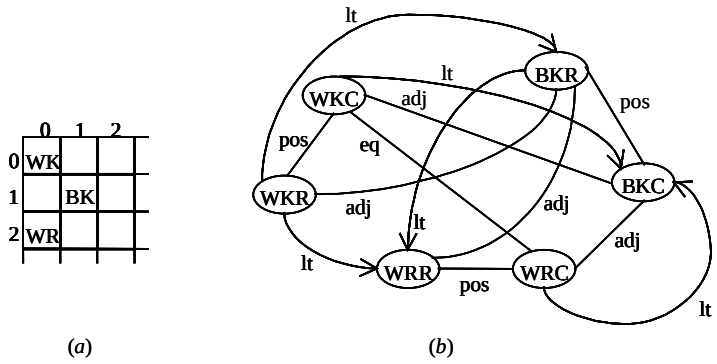

Figure 10: An example from the chess domain: (a) Board configuration, (b) SUBDUECL's graphical representation of the example.

Figure 10b shows SUBDUECL's representation for the chess domain example in Figure 10a. Figure 10a shows a corner of the chess board and the positions of the white king (WK), white rook (WR), and black king (BK). SUBDUECL's representation in Figure 10b uses two vertices for the row and column for each piece: white king row (WKR), white king column (WKC), white rook row (WRR), white rook column (WRC), black king row (BKR), and black king column (BKC). The row and column vertices for each piece are connected by an undirected edge labeled "pos" representing the row/column position relationship. These row and column vertices are then connected by various numeric relationship edges, depending on whether they are less than (lt), equal to (eq), or adjacent (adj).

Due to computational constraints only a subset (5,000 examples) of the entire database was used for a 10-fold cross validation. This experiment involved partitioning the data into 10 subsets of 500 each. Ten trials are executed using one of the partitions as a test set evaluated on the patterns learned using the other 4,500 examples as training. The ten accuracy results are then averaged to arrive at a final accuracy value. The accuracy results are 99.74% for PROGOL, 99.34% for FOIL, and 99.74% for SUBDUECL. FOIL and SUBDUECL perform significantly better than PROGOL. In terms of number of rules, PROGOL learned 6 rules, FOIL learned 11 rules, and SUBDUECL learned 7 rules (substructures).

The rules learned by PROGOL and FOIL are shown below, where the arguments of the predicate illegal are illegal(WKC,WKR,WRC,WRR,BKC,BKR). Rule number 1 identifies the illegal board configurations where the black

king and white king are in an adjacent row. Rule number two identifies the illegal board configurations where the white king and black king are in adjacent columns and rows. Rule three describes board configurations where the white king is in the same position as the white rook. Rule 4 says that board configurations where the white rook column is the same as the black king column are illegal. Rule 5 says that board configurations where the white rook row is the same as the black king row are illegal.

1. `illegal(A,B,C,D,A,E) :- adj(E,B).`
2. `illegal(A,B,C,D,E,F) :- adj(A,E), adj(B,F).`
3. `illegal(A,B,A,B,C,D).`
4. `illegal(A,B,C,D,C,E).`
5. `illegal(A,B,C,D,E,D).`

These rules say that a board configuration is illegal when:

1. The black king and white king are in an adjacent row.
2. The white king and black king are in adjacent columns and rows.
3. The white king is in the same position than the white rook.
4. The white rook column is the same as the black king column.
5. The white rook row is the same as the black king row.

The rules produced by FOIL are shown below.

1. `illegal(A,B,C,D,E,D) :- lt(A,C).`
2. `illegal(A,B,C,D,C,F) :- lt(A,C).`
3. `illegal(A,B,C,D,E,F) :- adj(A,E), adj(B,F).`
4. `illegal(A,B,A,D,E,D).`
5. `illegal(A,B,C,D,C,F) :- lt(C,A).`
6. `illegal(A,B,A,B,E,F).`
7. `illegal(A,B,A,D,A,F) :- adj(D,F).`

8. `illegal(A,B,C,D,E,D) :- illegal(C,G,D,G,H,B).`

9. `illegal(A,B,A,D,A,F) :- adj(A,B), illegal(A,A,A,D,G,B).`

10. `illegal(A,B,C,D,E,D) :- illegal(A,C,B,G,E,A).`

11. `illegal(A,B,A,D,A,F) :- lt(D,B), illegal(A,F,A,F,G,A).`

Rules 8-11 are recursive and not relevant to the domain. Rules 1-7 say that a board configuration is illegal when:

1. The white rook and the black king are in the same row, and the white king is in a lower column than the white rook.

2. The white rook and the black king are in the same column, and the white king is in a lower column than the white rook.

3. This rule is the same as rule 2 found by PROGOL.

4. The white king and the white rook are in the same column, and the white rook and the black king are in the same row.

5. The white rook and the black king are in the same column, and the white rook is in a lower column than the white king.

6. This rule is the same as rule 3 found by PROGOL.

7. The white king, the white rook and the black king are in the same column.

SUBDUECL found the equivalent substructures for rules number 3, 4, and 5 of PROGOL. For rule number 2 found by PROGOL, SUBDUECL found two rules where the only difference between them is the direction of one edge. Figure 11 shows the last two of the seven substructures found by SUBDUECL in the chess domain. These two substructures found by SUBDUECL's are not directly equivalent to any of the rules found by PROGOL, but are similar to PROGOL's rule 1.

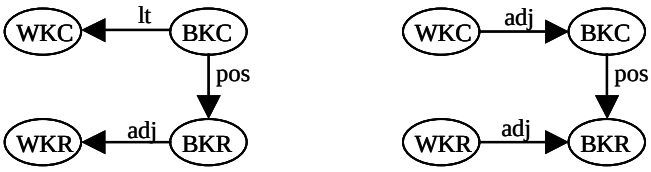

Figure 11: Two of seven substructures found by SUBDUECL in the chess domain.

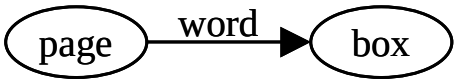

Figure 12: A substructure found in the web domain (Professors - Students).

5.2 Application to the World-Wide Web

The Web domain consists of graphs created from web sites. We have three options for the information that these graphs contain: hyperlink structure, hyperlink structure + page's titles, and hyperlink structure + page's content. We used a Perl program to extract this information and convert it into its graph representation. The first experiment that we made for the Web domain consisted in differentiating the Web pages of professors and students. We made the professors' web pages our positive examples and the students web pages our negative examples. For this initial experiment we chose the web page structure + page's content option, because we wanted to give as much information as possible to SUBDUECL and observe its behavior. SUBDUECL was able to find a very small substructure that could differentiate the positive examples from the negative examples. This substructure is shown in Figure 12 and says that every professor has in their web page the word "box", but students do not have this word in their web pages. This is true because the word "box" is part of the address field of the professors' web pages.

For the second experiment in the web domain we used the hyperlink structure option. We chose this option, because we wanted to learn only structure without considering the web pages' content so that SUBDUECL did not learn a substructure with a word (or set of words) describing the domain as in the previous experiment. We chose the structure graphs of web sites of computer stores as our positive examples and the structure

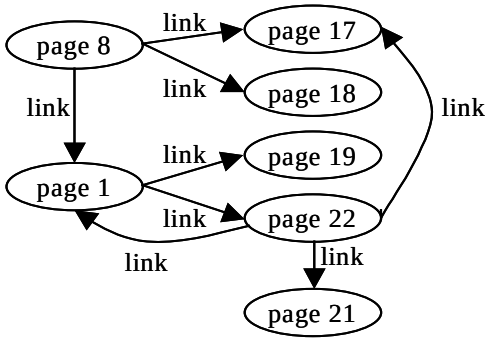

Figure 13: A substructure found in the web domain (Computer Stores - Professors).

graphs of web sites of professors as our negative examples. Figure 13 shows a substructure found in this domain. This substructure covered 24 of a total of 29 positive examples without covering any of the negative examples. Although not all the examples covered by this substructure had the same interpretation, for most of them the following explanation applies: Node 8 represents a category of products linked to three subcategories of products represented by nodes 1, 17, and 18. Nodes 19 and 22 represent either more specific subcategories derived from the subcategory represented by node 1 or specific products. Node 21 represents a specialization of the subcategory represented by node 22.

The results in the web domain tell us that SUBDUECL is able to learn how to differentiate between the graph structure of different classes of web sites. Further experimentation has been done in both artificial and real domains to show that SUBDUECL is competitive with ILP systems [18].

6 Conclusions

Current databases are maintaining and increasing amount of structural information, and older databases can be augmented with such structural information (e.g., spatial and temporal relationships). There is a need for advanced pattern learning algorithms that can handle structural data. We described our graph-based discovery approach to this task, and its implementation in the SUBDUE system. Iterative executions of SUBDUE yield a

hierarchical conceptual clustering of the data. Slight changes in the evaluation measure and top-level algorithm of SUBDUE yields SUBDUECL, a graph-based relational concept learner. These three functionalities (discovery, clustering, and concept learning) were illustrated using both artificial and real-world databases. Our conclusion is that the graph-based approach is both efficient and effective in identifying patterns in structural data. The approach is also competitive with older conceptual clustering techniques and inductive logic programming (ILP) approaches to relational concept learning.

We are pursuing several directions for enhancing the graph-based approach. Some of the enhancements are based on three advantages ILP systems have over SUBDUE. First, ILP systems can use variables to equate two arguments without specifying their value. Second, ILP systems can learn recursive concepts. Third, ILP systems are better at handling continuous values using inequalities and ranges. We are working on approaches to these three capabilities. We are working on introducing variables into a substructure definition based on the recognition of instances that have similar, but varying structure. We are moving to a graph grammar representation of substructures, clusters and concepts that will allow recursion. Lastly, we are investigating methods for learning ranges and distributions over continuous-valued labels in the graph. We believe these enhancements will give SUBDUE a clear advantage over ILP systems.

Finally, we will continue our empirical and theoretical analysis of SUBDUE. An increasing number of domains are becoming available that require structural pattern recognition to find desired patterns, e.g., domains in which relationships among a group of people might indicate suspicious behavior of the group. We have developed several parallel and distributed versions of SUBDUE to address very large databases, while maintaining the quality of the result [13]. We are also continuing our work in chemical domains (e.g., proteins, DNA, and carcinogens). We have begun a theoretical analysis of graph-based learning based on a probably-approximately correct (PAC) analysis [19] of conceptual graphs [31] and an analysis of the Galois lattice [21].

Our graph-based approach to discovery, clustering and concept learning provides an efficient, effective, and uniform approach to the task of extracting knowledge from structural data. We will continue to investigate improvements to the algorithms and implementations, as well as apply our approach to various problems. Source code and data for the SUBDUE system and the aforementioned extensions are available at http://cygnus.uta.edu/subdue.

7 Acknowledgements

This work is supported by NSF grant ILI-9615272. This effort is also sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-01-2-0570.

References

[1] G. H. Ball. Classification analysis. Technical Report SRI Project 5533, Stanford Research Institute, 1971.

[2] J. M. Barnard. Substructure searching methods: Old and new. *Journal of Chemical Information and Computing Sciences*, 33:532–538, 1993.

[3] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[4] H. Bunke and B. T. Messmer. A new algorithm for efficient subgraph matching. In G. Vernazza, A. N. Vebetsanopoulos, and C. Braccini, editors, *Image Processing: Theory and Applications*, pages 303–307. Elsevier Science Publishers, 1993.

[5] R. M. Cameron-Jones and J. R. Quinlan. Efficient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.

[6] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. MIT Press, 1996.

[7] 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.

[8] D. Conklin and J. Glasgow. Spatial analogy and subsumption. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 111–116, 1992.

[9] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[10] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[11] D. J. Cook, L. B. Holder, and S. Djoko. Knowledge discovery from structural data. *Journal of Intelligence and Information Sciences*, 5(3):229–245, 1995.

[12] D. J. Cook, L. B. Holder, and S. Djoko. Scalable discovery of informative structural concepts using domain knowledge. *IEEE Expert*, 11(5):59–68, 1996.

[13] D. J. Cook, L. B. Holder, G. Galal, and R. K. Maglothin. Approaches to parallel graph-based knowledge discovery. *Journal of Parallel and Distributed Computing*, 61(3):427–446, 2001.

[14] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[15] S. Dzeroski. Inductive logic programming and knowledge discovery in databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 5, pages 117–152. MIT Press, 1996.

[16] J. T. Favata. Offline general handwritten word recognition using an approximate beam matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9), 2001.

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

[18] J. Gonzalez. *Empirical and Theoretical Analysis of Relational Concept Learning Using a Graph-Based Representation.* PhD thesis, Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, Aug. 2001.

[19] P. Jappy and R. Nock. Pac learning conceptual graphs. In *Proceedings of the Sixth International Conference on Conceptual Structures*, 1998.

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

[21] M. Liquiere and J. Sallantin. Structural machine learning with galois lattice and graphs. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 305–313, 1998.

[22] J. Llados, E. Marti, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, 2001.

[23] B. Luo and E. R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.

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

[25] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[26] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

[27] L. D. Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1058–1063, 1993.

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

[29] J. Segen. Learning graph models of shape. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 29–35, 1988.

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

[31] J. F. Sowa. *Conceptual Structures: Information in Mind and Machine.* Addison Wesley, 1984.

[32] S. Su, D. J. Cook, and L. B. Holder. Applications of knowledge discovery to molecular biology: Identifying structural regularities in proteins.

In *Proceedings of the Pacific Symposium on Biocomputing*, pages 190–201, 1999.

[33] 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, 1991.

[34] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):889–895, 1998.

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

[36] P. H. Winston. *Artificial Intelligence*. Addison Wesley, 2nd edition, 1994.

[37] 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.