

Graph-Based Data Mining *

Diane J. Cook and Lawrence B. Holder
Department of Computer Science Engineering
University of Texas at Arlington
Email: {cook, holder}@cse.uta.edu
<http://cygnus.uta.edu/subdue.html>

Abstract

With the increasing amount and complexity of today's data, there is an urgent need to accelerate data mining of large databases. Furthermore, much of the data is structural in nature, or is composed of parts and relations between the parts. Hence, there exists a need to develop scalable tools to analyze and discover concepts in structural databases.

The goal of this research is to provide a system that performs data mining on databases represented as graphs. In this paper, we demonstrate how the SUBDUE system can be used to perform two key techniques for data mining: unsupervised pattern discovery and supervised concept learning from examples. Applications of the SUBDUE system to databases representing CAD circuits and protein molecules as well as to several databases from the UC Irvine repository will be presented to demonstrate the scalability and effectiveness of the graph-based system.

1 Introduction

The large amount of data collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns. In response to this problem, a number of researchers have developed techniques for discovering concepts in databases [3, 11, 16]. Although much of the data collected today has an explicit or implicit structural component (e.g., spatial or temporal), few discovery systems are designed to handle this type of data [10].

One method for discovering knowledge in structural data is the identification of common substructures (concept represented as graphs) 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

*Supported by NSF grant IRI-9615272 and THECB grant 003656-045.

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.

We describe the SUBDUE system that discovers interesting substructures in structural data. SUBDUE discovers substructures that compress the original database and represent interesting structural concepts in the data. By compressing previously-discovered substructures in the data, multiple passes of SUBDUE produce a hierarchical description of the structural regularities in the data.

In addition to performing unsupervised discovery of concepts in structural databases, SUBDUE can also induce concept descriptions from pre-classified data. The concept-learning version of SUBDUE accepts both a positive and negative graph and searches for a subgraph that compresses the positive graph, but not the negative graph.

In this paper, we provide an overview of the supervised and unsupervised learning capabilities of the SUBDUE system, and demonstrate the scalability and effectiveness of these data mining techniques on a variety of structural databases.

2 Related Work

A variety of approaches to unsupervised discovery using structural data have been proposed (e.g., [4, 21]). 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 observed data. While the above methods represent examples as distinct objects and process individual objects one at a time, our method stores the entire database (with embedded objects) as one graph and processes the graph as a whole.

Scientific discovery systems that use domain knowledge have also been developed. However, these systems are targeted for a single application domain. One example is MECHEM [22], which relies on domain knowledge to constrain the discovery of credible explanatory hypotheses specific to the domain of chemistry. In contrast, SUBDUE is devised for general-purpose automated discovery

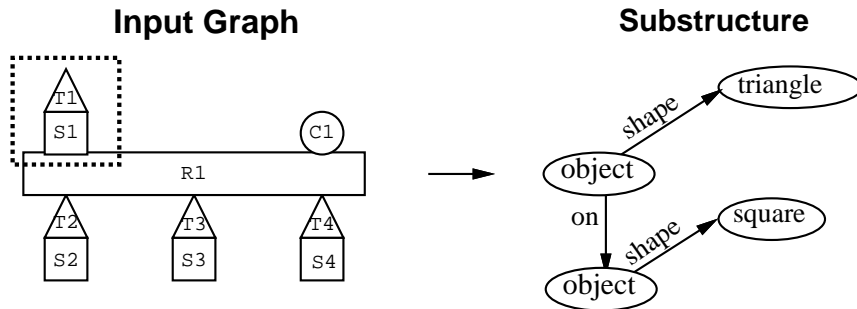


Figure 1: Example substructure in graph form.

with or without domain knowledge. Hence, the method can be applied to many structural domains.

Logic-based systems have dominated the area of relational concept learning, especially Inductive Logic Programming (ILP) systems. However, first-order logic can also be represented 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 larger hypothesis space.

An example ILP system studied in this paper is the FOIL system [2]. FOIL executes a top-down approach to learning relational concepts (theories) represented as an ordered sequence of function-free definite clauses. Given extensional background knowledge including relations and examples of the target concept relation, FOIL begins with the most general theory and follows a set-covering approach, which repeatedly adds a clause to the theory that covers some of the positive examples and few negative examples. Once such a clause is added to the theory, FOIL removes those positive examples covered by the clause and iterates on the reduced set of positive examples and all negative examples, until the theory covers all the positive examples. In order to avoid overly-complex clauses, FOIL ensures that the description length of the clause does not exceed the description length of the examples covered by the clause. In addition to the domains discussed here and numerous recursive and non-recursive logical domains, FOIL has been applied to learning search-control rules and learning patterns in hypertext domains.

3 Unsupervised Concept Discovery Using Subdue

SUBDUE discovers substructures that compress the original data and represent structural concepts in the data. The substructure discovery system 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. 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 representation of the substructure. This graphical representation serves as input to the substructure discovery system. Figure 1 shows a geometric example of a database. The graph representation of the discovered substructure is also shown, and one of the four instances of the substructure is highlighted in the picture.

The substructure discovery algorithm used by SUBDUE is a beam search. SUBDUE’s discovery algorithm is shown in figure 2. The first step of the algorithm is to initialize ParentList (containing substructures to be expanded), ChildList (containing substructures that have been expanded), and BestList (containing the highest-valued substructures SUBDUE has found so far) to be empty, and to set ProcessedSubs (the number of substructures that have been expanded so far) to 0. Each of the lists is a linked list of substructures, sorted in nonincreasing order by substructure value. For each unique vertex label, a substructure is assembled whose definition is a vertex with that label, and whose instances are all of the vertices in G with that label. Each of these substructures is inserted in ParentList.

The inner *while* loop is the core of the algorithm. Each substructure in turn is removed from the head of ParentList, and each of its instances is extended in all possible ways. This is done by adding a new edge and vertex in G to the instance, or just adding a new edge between two vertices, if both of the vertices are already part of the instance. The first instance of each unique expansion becomes a definition for a new child substructure, and all of the child instances that were expanded in the same way (i.e., by adding the same new edge or new edge with new vertex to the same old vertex) become instances of that child substructure. In addition, child instances that were generated by different expansions, and that match the child substructure definition within the matchcost threshold (see section 3.2), also become instances of the child substructure. Each child

```

SUBDUE( Graph, BeamWidth, MaxBest, MaxSubSize, Limit )
  ParentList = {}
  ChildList = {}
  BestList = {}
  ProcessedSubs = 0
  Create a substructure from each unique vertex label and its single-vertex
    instances; insert the resulting substructures in ParentList
  while ProcessedSubs <= Limit and ParentList is not empty do
    while ParentList is not empty do
      Parent = RemoveHead( ParentList)
      Extend each instance of Parent in all possible ways
      Group the extended instances into Child substructures
      foreach Child do
        if SizeOf( Child ) <= MaxSubSize then
          Evaluate the Child
          Insert Child in ChildList in order by value
          if Length( ChildList ) > BeamWidth then
            Destroy the substructure at the end of ChildList
        ProcessedSubs = ProcessedSubs + 1
        Insert Parent in BestList in order by value
        if Length( BestList ) > MaxBest then
          Destroy the substructure at the end of BestList
      Switch ParentList and ChildList
  return BestList

```

Figure 2: Subdue's discovery algorithm.

is then evaluated using the Minimum Description Length heuristic (see section 3.1), and inserted in ChildList in order by the heuristic value. The beam width of the search is enforced by controlling the length of ChildList: after inserting a new child into ChildList, if the length of ChildList exceeds the BeamWidth, the substructure at the end of the list is destroyed. The parent substructure is inserted in BestList; the same pruning mechanism is used to limit the length of BestList to be no greater than MaxBest. When ParentList has been emptied, ParentList and ChildList are switched, so that ParentList now holds the next generation of substructures to be expanded. SUBDUE's running time is constrained to be polynomial by the BeamWidth and Limit (a user-defined limit on the number of substructures to process) parameters, as well as by the computational constraints on the inexact graph match algorithm (see section 3.2).

One of Subdue's user-specified parameters controls whether the substructure discovery search space is pruned by discarding a child substructure whose heuristic value is not greater than its parent's heuristic value. Early in the discovery process, the number of instances of a given substructure is very large, and it dominates the value of any heuristic which uses the number of instances as a parameter. As the substructure grows, its number of instances decreases quickly, since the substructure is becoming more specific. This means that the heuristic value usually also decreases early in the discovery process. If it decreases for long enough, all the child substructures will be discarded, the list of substructures waiting for expansion will empty, and the discovery will halt. Disabling pruning during discovery keeps the child list full, even for substructure values that do not increase monotonically as the substructure definition grows in size. Since the list of substructures waiting for expansion never empties, another method for halting the program is needed. If the user specifies a maximum substructure size, child substructures that are larger than MaxSubSize will not be inserted in ChildList. This will cause ParentList and ChildList to eventually empty, and the discovery algorithm will halt.

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 in terms of the discovered substructures. This hierarchy provides varying levels of interpretation that can be accessed based on the specific goals of the data analysis.

3.1 The MDL Heuristic

Subdue’s heuristic for evaluating substructures is based on the Minimum Description Length (MDL) principle, which states that the best theory to describe a set of data is that theory which minimizes the description length of the entire data set [19]. Description length calculation is based on the model of a local computer, the encoder, sending a description of a concept to a remote computer, the decoder. The local computer must encode the concept as a string of bits that can be sent to the remote computer, which decodes the bit string to restore the original concept. The concept’s description length is the number of bits in the bit string. The MDL principle has been used for decision tree induction [17], pattern discovery in biosequences [1], image processing [13, 15, 14], concept learning from relational data [8], and learning models of non-homogeneous engineering domains [18].

SUBDUE’s implementation of the MDL principle is in the context of graph compression using a substructure. Here, the best substructure in a graph is one that minimizes $DL(S) + DL(G|S)$, where S is the discovered substructure, G is the input graph, $DL(S)$ is the number of bits required to encode the discovered substructure, and $DL(G|S)$ is the number of bits required to encode the input graph G after it has been compressed using substructure S . Cook and Holder [5] describe the exact computation of graph description length used in SUBDUE.

3.2 Inexact graph match

Because instances of a substructure can appear in different forms throughout the database, an inexact graph match is used to identify substructure instances [6]. 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 distortion costs can be determined by the user to bias the match for or against particular types of distortions.

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 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 search through the space of possible partial mappings using a uniform cost search. The cost from the root of the tree to a given node is calculated as the cost of all of the distortions corresponding to the partial mapping for that

node. Vertices from the matched graphs are considered in order from the most heavily connected vertex to the least connected. Because uniform cost search guarantees an optimal solution, the search ends as soon as the first complete mapping is found.

In addition, the user can limit the number of search nodes considered by the branch-and-bound procedure (defined as a function of the input graph sizes). 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 node at a given level. A complete description of the polynomial inexact graph match used by SUBDUE is provided by Cook and Holder [5].

Employing computational constraints such as a bound on the number of substructures considered (L) and the number of partial mappings considered during an inexact graph match (g), 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 $(\sum_{i=1}^L i * ((v - 1) - (i - 1))) * (v(L - 1)) * g$, where v represents the number of vertices in the input graph. The derivation of this expression is provided in the literature [6].

4 Applications of Unsupervised Subdue

The SUBDUE discovery system has been applied to databases in a number of domains. With each new application area, capabilities are identified that need to be added to our system. We have successfully applied SUBDUE with and without domain knowledge to databases in domains including image analysis, CAD circuit analysis, Chinese character databases, program source code, and chemical reaction chains [6, 9].

Figure 3 shows the substructures discovered from a CAD circuit representing a sixth-order bandpass “leapfrog” ladder (substructures inside boxes were discovered in previous iterations). The substructures are evaluated based on compression obtained using the substructure, time required to process the database, a human rating of the interestingness of the discovered concepts, and the number of substructure instances found in the database. Eight domain experts rating each substructure on a scale from 1 to 5, where 1 means the discovered substructures do not represent useful information in the domain and 5 means the discovered substructures are very useful.

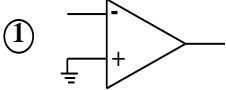
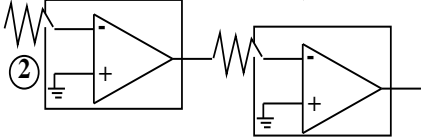
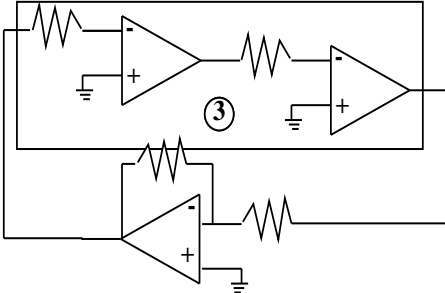
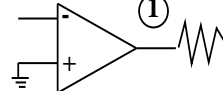
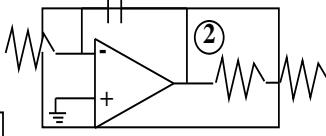
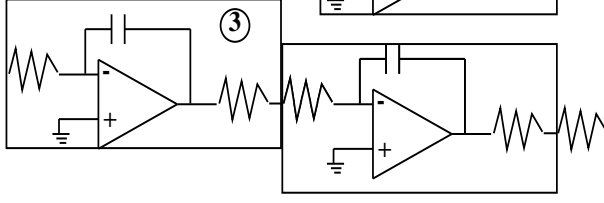
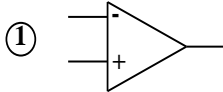
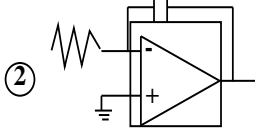
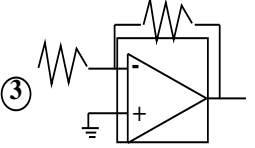
Usage of domain knowledge	Discovered Substructures	Compression	Nodes Expanded	Human Rating [std dev]	Instances
no domain knowledge		0.63	571,370	4.2 [1.2]	9
		0.68	46,422	2.7 [1.2]	3
		0.72	59,886	2.7 [1.0]	2
graph match rules		0.43	145,175	2.7 [1.7]	6
		0.51	39,881	2.7 [1.0]	3
		0.62	425,772	1.5 [0.8]	2
model knowledge and graph match rules		0.53	24,273	4.3 [1.2]	9
		0.66	12,960	4.5 [0.8]	4
		0.72	7,432	4.5 [0.8]	2

Figure 3: Leapfrog circuit – discovered substructures.

As the table shows, SUBDUE discovered substructures that performed well at compressing the database and represented functional concepts in the CAD domain. Using background knowledge in general improved the functional rating of the concept, but the MDL principal alone was effective in discovering the operational amplifier substructure, which was determined to be highly interesting and functional in this domain.

More recently, we have applied SUBDUE to several large databases that contain data whose interpretation will be beneficial to scientists, and that reveal capabilities that need to be added to the discovery system. First, we have applied SUBDUE to the July 1997 release of the Brookhaven Protein Data Bank (PDB). The goal was to identify structural patterns in primary, secondary, and tertiary structure of three categories of proteins: hemoglobin, myoglobin, and ribonuclease A. These patterns would act as signatures for the protein category, distinguishing proteins in the category from other types of proteins and providing a mechanism for classifying unknown proteins.

We convert primary structure information from the SEQRES records of each PDB file by representing each amino acid in the sequence as a graph vertex. The vertex numbers increase in the sequence order from N-terminus to C-terminus, and the vertex label is the name of the amino acid. An edge labeled “bond” is added between adjacent amino acids in a sequence.

Secondary structure is extracted by listing occurrences of helices and strands along the primary sequence. Each helix is represented by a graph vertex labeled “h” followed by the helix type and length. Each strand is represented by a graph vertex labeled “s” followed by the orientation of the strand and the length of the strand. Edges between two consecutive vertices is labeled “sh” if they belong to the same PDB file. The three-dimensional features of the protein are represented using the X, Y, and Z coordinates of each atom in the protein. Each amino acid α -carbon is represented as a graph vertex. If the distance between two α -carbons is greater than 6 Å, the information is discarded. Otherwise, edges between two α -carbons are created and labeled as “vs” (very short, distance ≤ 4 Å), or “s” (short).

SUBDUE indeed found such a pattern for each protein category. Using primary structure information, patterns were identified that were unique to each class of protein but occurred in 63 of the 65 hemoglobin cases, 67 of the 103 myoglobin cases, and 59 of the 68 ribonuclease A cases.

Figure 4 summarizes one of the findings for hemoglobin secondary structure, presenting an overall view of the protein, the portion of the protein where the SUBDUE-discovered pattern exists,

and the schematic views of the best pattern. The patterns discovered for each sample category covered a majority of the proteins in that category (33 of the 50 analyzed hemoglobin proteins, 67 of the 89 myoglobin proteins, and 35 of the 52 ribonuclease A proteins contained the discovered patterns). Detailed analysis of those that do not have the pattern indicates that there are many possible reasons. The structure of a protein is affected by many factors. The accuracy of the structure is affected by the quality of the protein sample, experimental conditions, and human error. Discrepancies may also be due to physiological and biochemical reasons. Structure of the same protein molecule may differ from one species to another. The protein may also be defective. For example, sickle-cell anemia is the classic example of a genetic hemoglobin disease. The defective protein does not have the right structure to perform its normal function.

The secondary structural patterns for the hemoglobine, myoglobin, and ribonuclease A proteins were mapped back into the PDB files. When mapped back, one discovered hemoglobin pattern was found to belong to the β chains and the other belonged to the α chains of a hemoglobin molecule (a hemoglobin molecular contains two α and two β chains). The discovered myoglobin pattern appears in a majority of the myoglobin proteins in the data set. Finally, upon mapping back discovered ribonuclease A patterns, we observed that several ribonuclease S proteins have the same patterns as those in ribonuclease A proteins. This is consistent with the fact that ribonuclease S is a complex consisting of two fragments (S-peptide and S-protein) of the ribonuclease A proteins. The pattern in the ribonuclease S comes from the S-protein fragment.

Dr. Steve Sprang, a molecular biologist at the University of Texas Southwestern Medical Center, evaluated the patterns discovered by the SUBDUE system. This scientist was asked to review the original database and the discovered substructures, and determine if the discovered concepts were indicative of the data and interesting discoveries. Dr. Sprang indicated that SUBDUE did find an interesting pattern in the data that was previously unknown and suggests new information about the micro-evolution of such proteins in mammals [20].

The secondary structure patterns discovered are also distinct to each protein category. The global data set is searched to identify the possible existence of the discovered pattern from each category. Results indicate that there is no exact match of the best patterns of one category in other category of proteins.

Application of the unsupervised and supervised learning algorithms in SUBDUE is continuing



Figure 4: Hemoglobin structure, discovered pattern, and schematic view.

in the domains of biochemistry, geology, program source code, and aviation data. SUBDUE brings unique capabilities to the analysis of data that is structural in nature.

5 Scalability of Subdue

One of the barriers to the integration of scientific discovery methods into practical data mining approaches is their lack of scalability. Many discovery systems are motivated from the desire to evaluate the correctness of a discovery method without regard to the method’s scalability. Another factor is that some scientific discovery systems deal with richer data representations that only degrade scalability. For example, SUBDUE’s discovery relies on computationally expensive procedures such as subgraph isomorphism. Although the algorithm has been polynomially constrained within SUBDUE, the system still spends a considerable amount of computation performing this task.

We are researching means of scaling SUBDUE using distributed hardware. In our approach, we partition the data among n individual processors and process each partition in parallel [12]. Each processor performs sequential SUBDUE on its local graph partition and broadcasts its best substructures to the other processors. The processors then evaluate the communicated substructures on their local partitions. Once all evaluations are complete, a master processor gathers the results and determines the global best discoveries.

The run time of sequential SUBDUE is nonlinear with respect to the size of the graph. As a result, decreasing the size of the input by partitioning the graph among multiple processors will sometimes result in speedup greater than the number of processors. However, the serial algorithm analyzes the graph in its entirety and will therefore not overlook important relationships between parts of the graph. Partitioning the graph among processors may remove essential information (in our case, edges along boundary lines), and neighboring information can no longer be used to

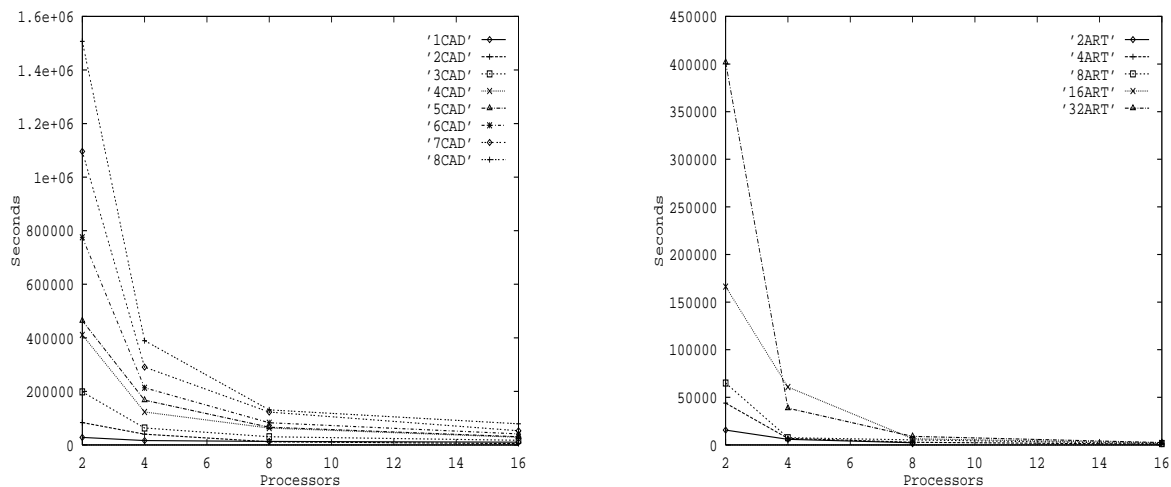


Figure 5: Distributed Subdue run time on CAD and ART graphs.

discover concepts.

The *Metis* graph partitioning algorithm (<http://www-users.cs.umn.edu/~karypis/metis>) is used to divide the input graph into n partitions in a way that minimizes the number of edges shared between partitions and thus reduces information loss. We modify this algorithm to allow a small amount of overlap between partitions, which recovers some of the information lost at the partition boundaries. Figure 5 graphs the run time of distributed SUBDUE on two classes of graphs as the number of processors increases. The two classes of graphs are graphs representing CAD circuits (each graph is labeled with “nCAD”, where n represents the size factor and 1CAD contains 8,441 vertices and 19,206 edges) and graphs generated artificially (each graph is labeled with “nART”, where n represents the size factor and 1ART contains 2,000 vertices and 5,000 edges). As predicted, the speedup is usually close to linear and sometimes greater than the number of processors. Increasing the number of partitions results in improved speedup until the number of partitions approaches the number of vertices in the graph.

Because each processor has only a portion of the overall database, some degradation in quality of discovered substructures might result from this parallel version of SUBDUE. In practice, we find that initially as the number of processors increases, quality measured as compression improves because a greater number of substructures can be considered in the allotted time. In the limit, as the number of partitions approaches the number of vertices in the graph, quality of discovered

substructures degrades.

By partitioning the database effectively, distributed SUBDUE proves to be a highly scalable system. One of our tested databases representing a satellite image contains 2 million vertices and 5 million edges and could not be effectively processed given the memory constraints on one machine, yet distributed SUBDUE processed the database in less than three hours using eight processors. The minimal amount of required communication, availability of distributed SUBDUE for wide-spread use, and use of PVM for communication allow the system to be run on a variety of heterogeneous distributed networks.

6 Supervised Concept Learning Using Subdue

We have extended SUBDUE to act not only as an unsupervised discovery system, but also to perform supervised graph-based relational concept learning. 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 learning routines, such as *covers* and *least general generalization*, which both require subgraph isomorphism tests in the general case. Learning methods for pattern recognition in chemical domains have been successful due to the natural graphical description of chemical compounds, but no domain-independent concept learning systems use a graph representation. Therefore, we have added a concept learning capability to the SUBDUE graph-based discovery system.

The main challenge in adding concept-learning capabilities to SUBDUE is the inclusion of a “negative” graph into the process. Substructures that occur often in the positive graph, but not often in the negative graph, are likely to represent the target concept. Therefore, the SUBDUE concept learner (which we will refer to as SUBDUECL) accepts both a positive and negative graph, and evaluates substructures based on their compression of the positive graph and lack of compression of the negative graph. SUBDUECL searches for the substructure S minimizing the cost of describing substructure S in positive graph G_p and negative graph G_n . This cost is expressed as

$$value(G_p, G_n, S) = DL(G_p, S) + DL(S) + DL(G_n) - DL(G_n, S)$$

where $DL(G, S)$ is the description length, according to the MDL encoding, of a graph G after being compressed using substructure S , and $DL(G)$ is the description length of a graph G . This cost

represents the information needed to represent the positive graph G_p using the substructure S plus the information needed to represent the portion of the negative graph G_n that was compressed using substructure S . Therefore, SUBDUECL prefers substructures that compress the positive graph, but not the negative graph.

A challenge in adding concept learning capabilities to SUBDUE is the discovery system’s bias toward finding only one good substructure in the entire input graph. ILP systems had a distinct advantage over SUBDUE, because they typically found theories composed of many rules; whereas, SUBDUE finds essentially one rule. The iterative, hierarchical capabilities of SUBDUE somewhat address this problem, but the substructures found in later iterations are typically defined in terms of previously-discovered substructures, and are therefore only specializations of the earlier, more general rule. To avoid this tendency, SUBDUECL discards any substructure that contains substructures discovered during previous iterations. SUBDUECL iterates until no substructure can be found that compresses the positive graph more than the negative graph.

7 Comparisons

We compare SUBDUECL to the ILP system FOIL [2] and to the decision-tree induction system C4.5 [16]. First, we compare the relational learners SUBDUECL and FOIL on a simple artificial domain (*house*) to discuss qualitative differences. We then compare all three systems on three relational domains drawn from the UC Irvine repository.

Figure 6 depicts the two datasets of the *house* domain, *house1* and *house2*, whose target concept is “triangle on square”. *House1* contains the six examples on the left, and *house2* contains an additional two examples shown on the right. Each object has a shape and is related to other objects using the binary relation *on*. The results of running the two systems on *house1* are as follows.

FOIL(80%): :- house(A,B,C). [3/0]

FOIL(50%): house(A,B,C) :- shape(A,triangle). [0/1]

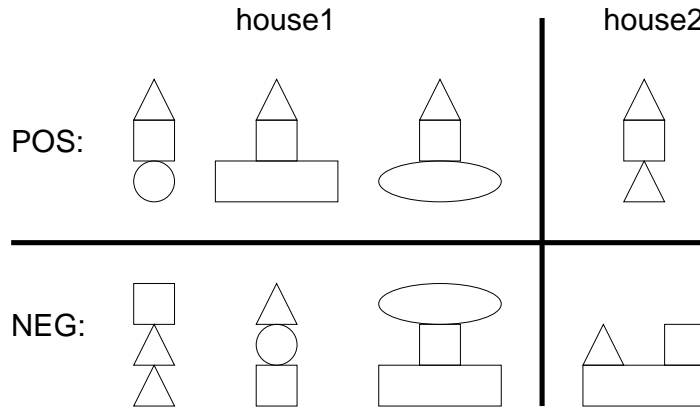
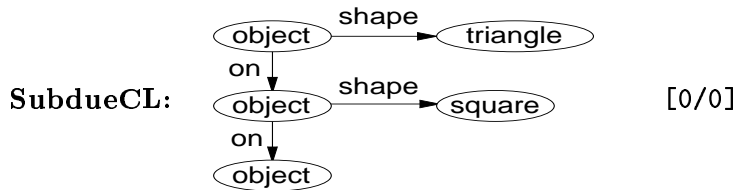


Figure 6: The house domain.



The bracketed numbers following the concepts are the number of false negatives and false positives. FOIL was run twice on each dataset, once with the default individual clause accuracy minimum of 80%, and once with 50%. At 80%, FOIL returned the concept that nothing was a house, misclassifying the three positive examples. At 50%, FOIL described houses as any example with a triangle on top. However, SUBDUECL identifies that the triangle should be *on* the square, because in order to relate the triangle and square objects in a connected substructure, SUBDUECL must include the *on* relation.

The *house2* dataset adds two more examples to *house1* to help emphasize the need for the *on* relation. The results of running the three systems on *house2* are as follows. Again, FOIL has trouble identifying the correct concept.

FOIL(80%): `house(A,B,C) :- shape(A,triangle), shape(C,triangle).` [3/0]

FOIL(50%): `house(A,B,C) :- shape(A,triangle).` [0/2]

SubdueCL: (same as for *house1*)

These examples reveal an advantage of the graph representation over logic for such position

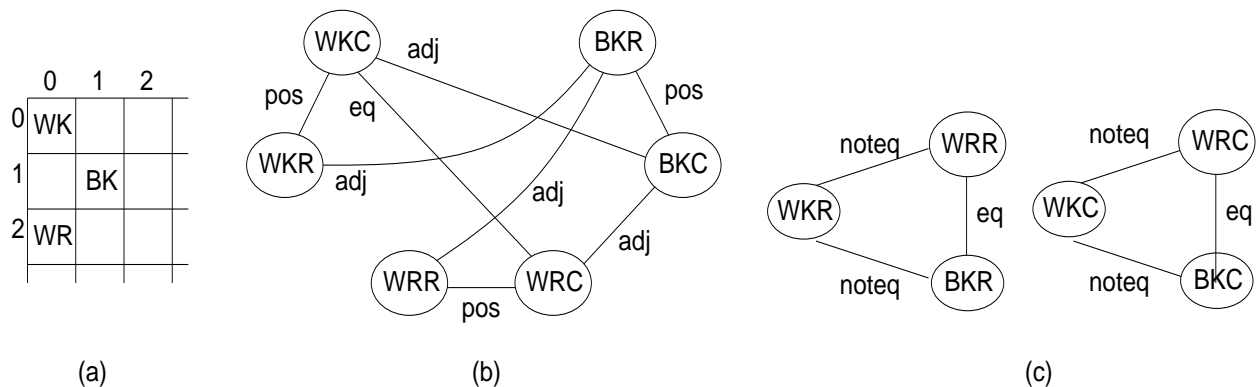


Figure 7: SubdueCL’s graphical representation (b) and discovered substructures (c) for the chess example in (a).

(and other relational) invariant concepts. However, there are concepts for which logic is better suited. For example, the concept of “top and bottom object have same shape” is easily represented as:

$\text{house}(A,B,C) :- \text{shape}(A,S), \text{shape}(C,S) .$ or

$\text{house}(A,B,C) :- \text{shape}(A,S1), \text{shape}(C,S2), S1 = S2 .$

The graph representation alluded to in the house domain would need to learn five different substructures, one for each of the five different shapes. Of course, if such a concept can be foreseen, then the graph representation could be changed to include an intermediate *shape* vertex connected to another vertex with the actual shape. Then, *equal* edges could be added between these *shape* vertices to represent equivalent shapes independent of the actual shape. Furthermore, although both systems handle numeric values (vertex labels), learning concepts involving general equalities and inequalities between numbers requires a similar representational transformation as for shape equality above.

The relational domains we have tested include illegal *chess* endgames, tic tac toe endgames, and Bach chorales. The chess domain consists of 12,957 row-column positions for a white king, white rook, and black king such that the black king is (positive) or is not (negative) in check. FOIL extensionally defines adjacency relations between chess-board positions and less-than relations between row and column numbers. We provide C4.5 with the same information by adding features that relate the row and column values of each piece as equal, not equal, or adjacent.

Figure 7 shows SUBDUECL’s representation for a *chess* domain example. Each piece is represented by two vertices corresponding to the row and column of the piece, connected by a *position* relation (e.g., WKC stands for white king column). Instead of *less-than* relations, we used *eq* and *noteq* relations between all such rows and columns.

All three systems were tested for predictive accuracy on this domain using three-fold cross validation, with significance values gathered using a paired student t-test. The accuracy results for this experiment are 99.80% for FOIL, 99.77% for C4.5 and 99.21% for SUBDUE. The difference in accuracies between FOIL and SUBDUE are significant at the 0.19 level (i.e., the probability that the difference is insignificant is 0.19), and the difference in accuracies between C4.5 and SUBDUE is significant at the 0.23 level. FOIL learned 6 rules, SUBDUECL learned 5 rules (substructures), and C4.5 learned 43 rules. Four rules were discovered by all of the systems that described in each case approximately 2,000 of the 2,118 positive examples (two of these rules described as substructures are shown in Figure 7 (c)). The remaining rules differed between each system, and these rules proved to be more powerful for FOIL and C4.5 because of the ability to learn numeric ranges.

Next, we tested the systems on a complete set of 958 possible board configurations at the end of tic-tac-toe games. The target concept is “a win for x”. All three systems are supplied with the (x,o,blank) values for each of the nine board positions. Unlike the other systems, SUBDUE does not key on individual position values but uses the relational information between board positions to learn the three concepts of three-in-a-row, three-in-a-column, and three-in-a-diagonal. The accuracy results are therefore 100.00% for SUBDUE, 92.35% for FOIL (the difference is significant at the 0.21 level), and 96.03% for C4.5 (the difference is significant at the 0.03 level).

In the final experiment we use a set of musical excerpts described by a sequence of pitch values to differentiate Beethoven works from Bach chorales. In each of the 50 Beethoven examples a musical theme from a piece of music is repeated in 10 examples, with a varying pitch offset and surrounded by random pitch values. The 100 Bach chorales are drawn from the UC Irvine repository. The systems were tested using absolute pitch values and using the relative difference between one pitch value and the next. All three systems performed better using pitch difference values. Unlike FOIL and C4.5, SUBDUE used the relational information between successive notes to learn the embedded musical sequences for the positive examples. The accuracy values are 100.00% for SUBDUE, 85.71% for FOIL (the difference is significant at the 0.06 level), and 82.00% for C4.5 (the difference is

significant at the 0.00 level).

8 Conclusions

The increasing structural component of today's databases requires data mining algorithms capable of handling structural information. The SUBDUE system is specifically designed to discover concepts in structural databases. In this paper, we have described how SUBDUE can be used in a supervised or unsupervised fashion to efficiently learn concepts in a variety of structural databases.

First, SUBDUE represents a method for integrating domain independent and domain dependent substructure discovery based on the minimum description length principle. The method is generally applicable to many structural databases, such as computer aided design (CAD) circuit data, molecular data, computer programs, etc. SUBDUE has been shown to effectively discover substructures relevant to the domain of study. The distributed implementation of SUBDUE yields scalability to large databases in terms of memory utilization and computational performance.

Second, we have introduced the graph-based relational concept learner SUBDUECL and compared it to the logic-based relational concept learner FOIL and to the decision-tree induction system C4.5. SUBDUECL accepts both a positive and negative graph, and searches for a subgraph that compresses the positive graph, but not the negative graph. Experimental results indicate that the graph-based concept learner can more-easily learn some structural concepts (e.g., a pattern among a subsequence of the arguments to the relation, but independent of the relative position of the subsequence within the ordered list of arguments). However, other concepts (e.g., equality constraints between arguments and recursion) are more-easily learned by logic-based concept learners. The quantitative results indicate that the graph-based relational concept learner is competitive with logic-based relational concept learners on a variety of domains.

This comparison has identified a number of avenues for enhancements to both types of learners. The graph-based learner SubdueCL would benefit from the ability to identify ranges of numbers. This could be accomplished by utilizing the system's existing capability to find similar but not exact matches of a substructure in the input graph. Numeric values within the instances could be generalized to the encompassing range. A graph-based learner also needs the ability to represent recursion, which plays a central part in many logic-based concepts. More research is needed to

identify enhancements to the representation that can describe recursive structures, such as using graph grammars. Future work will also focus on extending the SUBDUE system to handle other forms of learning such as clustering.

We are continuing work in the area of biochemistry by applying SUBDUE to data from the Human Genome Project and attempting to find patterns in the DNA sequence that would indicate the presence of a gene transcription factor site. Unlike other approaches to finding patterns in gene data (for example, [7]), we use a graph to represent structural information in the sequence. It is our hope that these discovered patterns will be indicative of genes in uncharted areas of the DNA sequence.

Also in the area of chemistry, we are applying SUBDUE to the Predictive Toxicology Challenge data. This data contains the structural descriptions of more than 300 chemical compounds that have been analyzed for their carcinogenicity. Each compound (except for about 30 held out for future testing) is labeled as either cancer-causing or not. Our goal is to find a pattern in the cancerous compounds that does not occur in the non-cancerous compounds. The concept-learning version of SUBDUE has found several promising patterns that distinguish the cancerous and non-cancerous compounds. These patterns are currently under evaluation by a faculty member in UTA's department of Chemistry.

In addition, we are applying SUBDUE to a number of other databases, including the Aviation Safety Reporting System (ASRS) database, U.S. Geological Survey Earthquake data, and software call graphs. SUBDUE has discovered several interesting patterns in the ASRS database. Dr. Burkart of the Geology department at UTA has evaluated SUBDUE's results on the geology data and stated that SUBDUE correctly identified patterns dependent on the depth of earthquakes, because depth is often the distinguishing factor between types of earthquakes. These and other results have successfully shown that Subdue discovers relevant knowledge in structural data, and scales to large databases.

References

- [1] A. Brazma, I. Jonassen, E. Ukkonen, and J. Vilo. Discovering patterns and subfamilies in biosequences. In *Proceedings of the Fourth International Conference on Intelligent Systems*

- for *Molecular Biology*, pages 34–43, 1996.
- [2] R. M. Cameron-Jones and J. R. Quinlan. Efficient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.
 - [3] 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.
 - [4] D. Conklin. Machine discovery of protein motifs. *Machine Learning*, 21:125–150, 1995.
 - [5] 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.
 - [6] D. J. Cook, L. B. Holder, and S. Djoko. Scalable discovery of informative structural concepts using domain knowledge. *IEEE Expert*, 11(5), 1996.
 - [7] M. W. Craven and J. W. Shavlik. Machine learning approaches to gene recognition. *IEEE Expert*, 9(2):2–10, 1993.
 - [8] M. Derthick. A minimal encoding approach to feature discovery. In *Proceedings of the National Conference on Artificial Intelligence*, pages 565–571, 1991.
 - [9] S. Djoko, D. J. Cook, and L. B. Holder. An empirical study of domain knowledge and its benefits to substructure discovery. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):575–586, 1997.
 - [10] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. MIT Press, 1996.
 - [11] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
 - [12] G. Galal, D. J. Cook, and L. B. Holder. Improving scalability in a scientific discovery system by exploiting parallelism. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 171–174, 1997.

- [13] Y. G. Leclerc. Constructing simple stable descriptions for image partitioning. *International journal of Computer Vision*, 3(1):73–102, 1989.
- [14] 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.
- [15] A. Pentland. Part segmentation for object recognition. *Neural Computation*, 1:82–91, 1989.
- [16] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [17] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.
- [18] R. B. Rao and S. C. Lu. Learning engineering models with the minimum description length principle. In *Proceedings of the National Conference on Artificial Intelligence*, pages 717–722, 1992.
- [19] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.
- [20] S. Sprang. Personal Communication, 1998.
- [21] 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.
- [22] R. E. Valdes-Perez. Conjecturing hidden entities by means of simplicity and conservation laws: Machine discovery in chemistry. *Artificial Intelligence*, 65:247–280, 1994.