Scalable Discovery Of Informative Structural Concepts Using Domain Knowledge *

Diane J. Cook, Lawrence B. Holder, and Surnjani Djoko
University of Texas at Arlington
Arlington, TX 76019
USA
{cook,holder,djoko}@cse.uta.edu

Abstract

Discovering repetitive, and functional substructures in large structural databases improves the ability to interpret and compress the data. However, scientists working with a database in their area of expertise often search for predetermined types of structures, or for structures exhibiting characteristics specific to the domain. This paper presents a method for guiding the discovery process with domain-specific knowledge. In this paper, the SUBDUE discovery system is used to evaluate the benefits of using domain knowledge to guide the discovery process. Results show that domain-specific knowledge improves the search for substructures which are useful to the domain, and leads to greater compression of the data. Empirical and theoretical results also indicate the scalability of the algorithm to increasingly large structural databases.

Keywords–data mining, minimum description length principle, data compression, inexact graph match, domain knowledge, scalability

*Supported by NASA grant NAS5-32337.

1 Introduction

With the increasing amount and complexity of today's data, there is an urgent need to accelerate discovery of knowledge in large databases. In response to this need, numerous approaches have been developed for discovering concepts in databases using a linear, attribute-value representation. These approaches address issues of data relevance, missing data, noise, and utilization of domain knowledge. However, much of the data that is collected 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 [5]. Many reported discovery tools are also computationally expensive and cannot scale easily to large databases, especially those containing structural information.

Recently, we introduced a method for discovering substructures in structural databases using the minimum description length (MDL) principle [3]. The system is called SUBDUE, and it 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 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].

Although the MDL principle is useful for discovering substructures that maximize compression of the data, scientists often employ knowledge or assumptions of a specific domain to guide the discovery process. A domain-independent discovery method is valuable in that the discovery of unexpected substructures is not blocked. However, the discovered substructures might not be useful to the user. On the other hand, using domain-specific knowledge can assist the discovery process by focusing search and can also help make the discovered substructures more meaningful to the user. Hence,

in order to trade off between domain-independent and domain-dependent discovery methods, we incorporate domain knowledge into the SUBDUE system, and combine both the domain-independent and domain-dependent methods to guide the search toward more appropriate substructures.

A variety of approaches to discovery using structural data have been proposed (e.g., [2, 6, 9, 10]). Many approaches use a knowledge base of concepts to classify the structural data. The purposes of the knowledge base in these systems are 1) to improve the performance of graph comparisons and retrieval, where the individual graphs are maintained in a partial ordering defined by the subgraph-of relation, 2) to deepen the hierarchical description, and 3) to group objects into more general concepts. These systems perform concept learning over examples and categorization of observed data. SUBDUE allows the use of both domain-independent heuristics and domain-dependent knowledge. While the above methods process individual objects one at a time, our method is designed to process the entire structural database, which consists of many objects.

This paper focuses on a method of realizing the benefits of domain-dependent discovery approaches by adding domain-specific knowledge to a domain-independent discovery system. Secondly, this paper explicitly evaluates the benefits and costs of utilizing domain-specific information. In particular, the performance of the SUBDUE system is measured with and without domain-specific knowledge along the performance dimensions of compression, time needed to discover the substructures, and usefulness of the discovered substructures. Thirdly, this paper addresses the issue of scalability of structure discovery using SUBDUE. Scalability tests are performed and features of databases that can affect the performance of SUBDUE are highlighted.

The following sections describe the approach in detail. Section 2 introduces needed definitions. Section 3 describes the minimum description length principle used by this approach, encoding scheme, and the discovery process, and Section 4 presents the inexact graph match algorithm employed by SUBDUE. Section 5 describes methods of incorporating domain knowledge into the substructure discovery process. The evaluations detailed in Section 6 demonstrate SUBDUE's ability to find substructures that

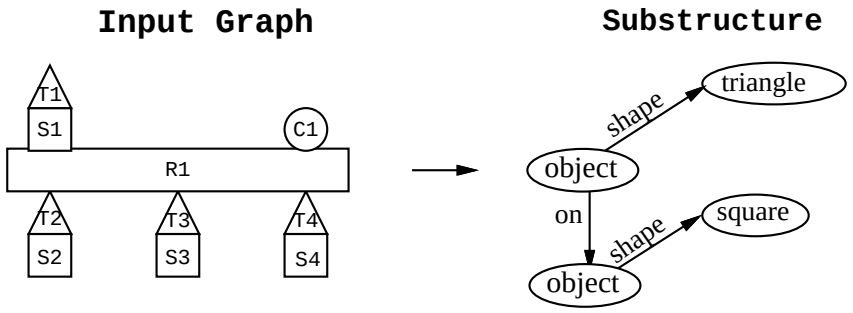

Figure 1: Example substructure in graph form.

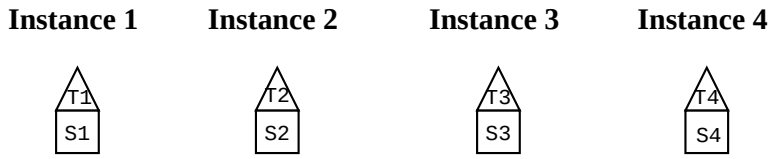

Figure 2: Instances of the substructure.

compress the data and to re-discover known concepts in a variety of domains. Section 7 provides an analysis of the run-time complexity of SUBDUE and an evaluation of SUBDUE's scalability.

2 Structural data representation

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. This graphical representation serves as input to the substructure discovery system. Figure 1 shows a geometric example of such an input graph. The objects in the figure (e.g., `T1`, `S1`, `R1`) become labeled vertices in the graph, and the relationships (e.g., `on(T1,S1)`, `shape(C1,circle)`) become labeled edges in the graph. The graphical representation of the substructure discovered by SUBDUE from this data is also shown in Figure 1.

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. For example, the instances of the substructure in Figure 1 are shown in Figure 2.

3 Substructure discovery using the MDL principle

The minimum description length (MDL) principle introduced by Rissanen [8] states that the best theory to describe a set of data is a theory which minimizes the description length of the entire data set. The MDL principle has been used for decision tree induction, image processing, concept learning from relational data, and learning models of non-homogeneous engineering domains.

We demonstrate how the minimum description length principle can be used to discover substructures in complex data. In particular, a substructure is evaluated based on how well it can compress the entire data set. We define the minimum description length of a graph to be the minimum number of bits necessary to completely describe the graph. SUBDUE searches for a substructure that minimizes $I(S) + I(G|S)$, where S is the discovered substructure, G is the input graph, $I(S)$ is the number of bits (description length) required to encode the discovered substructure, and $I(G|S)$ is the number of bits required to encode the input graph G with respect to S.

3.1 Graph encoding scheme

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

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

1. Determine the number of bits *vbits* needed to encode the vertex labels of the graph. First, we need $(\lg v)$ bits to encode the number of vertices v in the graph.

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

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

2. Determine the number of bits *rbits* needed to encode the rows of the adjacency matrix A. Typically, in large graphs, a single vertex has edges to only a small percentage of the vertices in the entire graph. Therefore, a typical row in the adjacency matrix will have much fewer than v 1s, where v is the total number of vertices in the graph. We apply a variant of the coding scheme used by [7] to encode bit strings with length n consisting of k 1s and $(n-k)$ 0s, where $k \ll (n-k)$. In our case, row i $(1 \leq i \leq v)$ can be represented as a bit string of length v containing k_i 1s. If we let $b = \max_i k_i$, then the i^{th} row of the adjacency matrix can be encoded as follows.

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

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

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

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

3. Determine the number of bits *ebits* needed to encode the edges represented by the entries $A[i,j] = 1$ of the adjacency matrix A. The number of bits needed to encode entry $A[i,j]$ is $(\lg m) + e(i,j)[1 + \lg l_u]$, where $e(i,j)$ is the actual

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

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

where e is the number of edges in the graph, and K is the number of 1s in the adjacency matrix.

3.2 Substructure discovery without domain knowledge

The substructure discovery algorithm used by SUBDUE is a computationally-constrained beam search. The algorithm begins with an initial set of substructures matching every distinctly-labeled vertex in the graph. Each iteration through the algorithm selects the best substructure according to its ability to minimize the description length of the entire graph, and expands the instances of the best substructure by one neighboring edge in all possible ways. The new unique generated substructures become candidates for further expansion. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit. The evaluation of each substructure is guided by the MDL principle.

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

To represent an input graph using a discovered substructure involves additional overhead to replace the substructure's instances with a pointer to the newly-discovered substructure. Therefore, the number of bits needed to represent G, given the discovered substructure S, is

$$\begin{aligned} I(G|S) &= I(G) - \sum_{i=1}^{n} I(S) + \sum_{i=1}^{n} I(pointer) \\ &= I(G) - nI(S) + nI(pointer), \end{aligned}$$

where n is the number of instances found for the discovered substructures. The second term is the sum of bits saved over the discovered substructure, and the last term is the sum of bits needed for the overhead.

We define a compression measure to evaluate a substructure's ability to compress an input graph as the following

$$Compression = 1 - (\frac{DL\ of\ compressed\ graph}{DL\ of\ original\ graph}),$$

where $DL\ of\ compressed\ graph$ is $I(G|S)+I(S)$, and $DL\ of\ original\ graph$ is $I(G)$. If $Compression$ is greater than zero, the representation of G using S is used instead of the original representation, since it required fewer bits.

Both the input graph and the discovered substructure can be encoded using the above encoding scheme. After a substructure is discovered, each instance of the substructure in the input graph is replaced by a single vertex representing the entire substructure.

4 Inexact graph match

The use of a graph as a representation for data and concepts requires methods for matching data to concepts. Methods of graph matching can be categorized into exact graph matching, and inexact matching based on graph distance or probability, transformation cost, graph identity, and minimal representation criterion.

Although exact structure match can be used to find many interesting substructures, many of the substructures show up in a slightly different form throughout the

data. These differences may be due to noise, distortion, or may just illustrate slight differences between instances of the same general class of structures.

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

In this inexact match approach, each distortion of a graph is assigned a cost. A distortion is described in terms of basic transformations such as deletion, insertion, and substitution of vertices and edges. The 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 extend Bunke's approach by adding a branch-and-bound search. The set of possible partial mappings can be viewed as a search tree. 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, as this constrains the remaining match. Because branch-and-bound 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. By defining such a limit, significant speedup can be realized at the expense of accuracy for the computed match cost. A complete description of the inexact graph match procedure used by SUBDUE is provided by

Cook and Holder [3].

5 Adding domain knowledge to the SUBDUE system

The SUBDUE discovery system was initially developed using only domain independent heuristics to evaluate potential substructures. As a result, some of the discovered substructures may not be useful and relevant to specific domains of interest. For instance, in a programming domain, the BEGIN and END statements may appear repetitively within a program; however, they do not perform any meaningful function on their own; hence they exhibit limited usefulness. Similarly, in the CAD circuit domain, some sub-circuits or substructures may appear repetitively within the data; however, they may not perform meaningful functions within the domain of usage. To make SUBDUE's discovered substructures more interesting and useful across a wide variety of domains, domain knowledge is added to guide the discovery process. Furthermore, compressing the graph using the domain knowledge can increase the chance of realizing greater compression than without using the domain knowledge.

In this section we present several types of domain knowledge that are used in the discovery process and explain how they bias discovery toward certain types of substructures.

5.1 Model/Structure knowledge

Model/Structure knowledge provides to the discovery system specific types of structures that are likely to exist in a database and that are of particular interest to a scientist using the system. The model knowledge is organized in a hierarchy that specifies the connection between individual structures. Nodes of the hierarchy can be classified as either primitive (non-decomposable) or non-primitive. The primitive nodes reside in the lowest level, i.e., the leaves, and all non-primitive nodes reside in the higher levels of the hierarchy. The primitive nodes represent basic elements of the

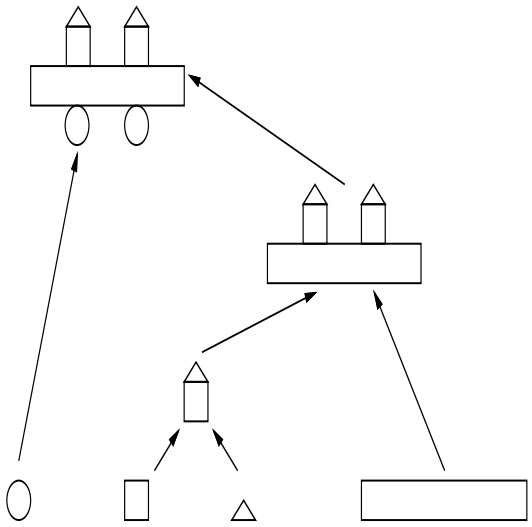

Figure 3: A simple model / structure hierarchy.

domain, whereas the non-primitive nodes represent models or structures which consist of a conglomeration of primitive nodes and/or lower-level non-primitive nodes. The higher the node's level, the more complex is the structure it represents. The hierarchy for a particular domain is supplied by a domain expert. The structures in the hierarchy and their functionalities are well known in the context of that domain. This knowledge is formed in a bottom-up fashion. Users can extend the hierarchy by adding new models.

To illustrate the structure knowledge, a simple example is shown in Figure 3, representing a hierarchy based on the shape structure. The primitive nodes are triangle, square, circle and rectangle. The non-primitive nodes are built upon the primitive nodes and/or non-primitive nodes. While Figure 3 represents a hierarchy built using commonalities between individuals' shape, in the programming and computer aided design (CAD) circuit domain, the hierarchies are built based on commonalities between individuals' functional structure. For example, in the CAD circuit domain, basic components of a circuit (e.g., resistor, transistor) are represented by primitive nodes, and functional sub-circuits such as operational amplifier, filter, etc. are represented by non-primitive nodes. This hierarchical representation allows examining of the structure knowledge at various levels of abstraction, focusing the search and

reducing the search space.

Although the minimum description length principle still drives the discovery process, domain knowledge is used to input a bias toward certain types of substructures. First, the modified version of SUBDUE can be biased to look specifically for structures of the type specified in the model hierarchy. The discovery process begins with matching a single vertex in the input graph to primitive nodes of the model knowledge hierarchy. If the primitive nodes do not match the input vertices, the higher level nodes of the hierarchy are pursued. The models in the hierarchy pointed to by the matched model nodes in the input graph are selected as candidate models to be matched with the input substructure. Each iteration through the process, SUBDUE selects a substructure from the input graph which has the best match to one of the selected models and can be used to compress the input graph. The match can either be a subgraph match or a whole graph match. If the match is a subgraph match, SUBDUE expands the instances of the best substructure by one neighboring edge in all possible ways. The newly generated substructure becomes a candidate for the next iteration. However, if the match is a whole graph match, the process has found the desired substructure, and the chosen substructure is used to compress the entire input graph. The process continues to expand the substructure until either a substructure has been found or all possible substructures have been considered.

To represent an input graph using a discovered substructure from the model hierarchy, the representation involves additional overhead to replace the substructure's instances with a pointer to the model hierarchy. In some cases, a model definition includes parameters which must also be represented. Consider an example in the programming domain where a substructure of the model hierarchy (e.g., *Sort(a, b)*), where a and b are dummy variables) is discovered in a program. SUBDUE replaces each of the discovered substructure's instances with $Sort(a_i, b_i)$, where *Sort* is a pointer to the model hierarchy, and a_i and b_i are parameters of the ith instance.

Therefore, the number of bits needed to represent G, given the substructure S

which matches the model M is

$$\begin{aligned} I(G|M) &= I(G) - \sum_{i=1}^{n} I(S) + \sum_{i=1}^{n} I(pointer) + \sum_{i=1}^{n} I(parameters_i) \\ &= I(G) - nI(S) + nI(pointer) + \sum_{i=1}^{n} I(parameters_i), \end{aligned}$$

where n is the number of instances found for the discovered substructure. The second term is the sum of the bits saved over the discovered substructure, and the last two terms are the sum of bits needed for the overhead.

When the substructure only matches part of a model (subgraph match), then representing the model includes an overhead associated with specifying the path to the model in the hierarchy and the mapping of all the substructure's vertices and edges to part of the model's vertices and edges. The mapping describes how many vertices of the model, how many edges of the model, and which vertices and edges of the model are matched to the substructure.

If the part of the model matching the substructure is too small, the savings may not cover the overhead cost. Consequently, when the match is a subgraph match, the number of bits needed to represent M is

$$I(M) = I(path) + I(mapping_v) + I(mapping_e),$$

where $I(mapping_v)$ is the number of bits needed to describe the mapping for the vertices, and $I(mapping_e)$ is the number of bits needed to describe the mapping for the edges.

However, when the substructure matches all parts of a model graph (whole graph match), there is no need to indicate the mapping, because we assume the mapping order of the substructure is the same as the order of model. Thus the overhead incurred includes only the path to the model in the hierarchy. When a whole graph match is found, the number of bits needed to represent M is

$$I(M) = I(path).$$

$I(path)$ is encoded as a path in the hierarchy of model knowledge, where a path begins at the matched primitive node and terminates at the found model.

$$I(path) = Level \times \lg l_h,$$

where $Level$ is the depth of the model in the hierarchy and l_h is the number of unique models in the hierarchy.

In the computation for subgraph match, $I(mapping_v)$ is encoded as the following:

$$I(mapping_v) = \lg nv_s + \lg \binom{nv_m}{nv_s},$$

where nv_s is the number of vertices in the substructure and nv_m is the number of vertices in the model. The first term describes the number of mapped vertices, and the second term describes which vertices are mapped.

Similarly, $I(mapping_e)$ is encoded as the following

$$I(mapping_e) = \lg ne_s + \lg \binom{ne_m}{ne_s},$$

where ne_s is the number of substructure's edges and ne_m is the number of model's edges. The first term describes how many edges are mapped, and the second term describes which edges are mapped.

If $Compression$ is greater than zero, the representation of G using S which matches the model M is used instead of the original representation. After a substructure is discovered, each instance of the discovered substructure in the input graph is replaced by a pointer to a predefined model in the model hierarchy which representing the substructure.

5.2 Graph match rules

At the heart of the SUBDUE system lies an inexact graph match algorithm that finds instances of a substructure definition. Since many of substructure instances can show up in a slightly different form throughout the data, and each of these differences is described in terms of basic transformations performed by the graph match, we can use graph match rules to assign each transformation a cost based on the domain of usage. This type of domain-specific information is represented using if-then rules such as the following:

IF (domain = x) and (perform graph match transformation y)

THEN (graph match cost = z)

To illustrate this rule, consider an example in the CAD circuit domain. In many representations of CAD circuits, unique instances of a resistor are given unique vertex labels in the graph. However, we allow a vertex representing one resistor (label begins with "R") to match to any other instance of a resistor. Similarly, in a programming domain we may allow a variable vertex to be substituted by another variable, but do not allow a vertex representing an special symbol or function call to be substituted by another vertex. A specific rule can then be represented as the following:

IF (domain = CAD) and ($Char_1$(Vertex1) = $Char_1$(Vertex2) = "R")
THEN graph match cost = 0.0;

ELSE IF (Vertex1 <> Vertex2)
THEN graph match cost = 2.0;

The graph match rules allow a specification of the amount of acceptable generality between a substructure definition and its instances, or between a model definition and its instances in the input graph. Given $g1$, $g2$, and a set of distortion costs, the actual calculation of similarity can be performed using the search procedure described earlier. As long as the similarity is within the user-defined threshold, the two graphs $g1$ and $g2$ are considered to be isomorphic.

6 Evaluation

In this section, we evaluate the benefits and costs of utilizing the domain-specific knowledge in performing substructure discovery. We measure the performance of SUBDUE with and without domain-specific knowledge when applied to databases in the CAD circuit and artificial domains. The goals of our substructure discovery system are to efficiently find substructures that can reduce the description length needed to describe the data, and to find substructures that are considered useful for the given domain.

To evaluate SUBDUE in the CAD circuit domains presented in Section 6.1, we compare SUBDUE's discovered substructures to human ratings. If the approach has some validity, SUBDUE should prefer substructures which were rated highly by humans. Three types of discovered substructures are evaluated: 1) substructures discovered without using the domain knowledge, 2) substructures discovered using the graph match rules, and 3) substructures discovered using a combination of model knowledge and graph match rules. The performance of the system is measured along three dimensions: 1) compression, which shows a substructure's ability to compress an input graph, 2) number of search nodes expanded by SUBDUE, which indicates the time to discover a substructure, and 3) average evaluation value and standard deviation of human rating, which give the interestingness of a substructure as measured by human experts. The interestingness of SUBDUE's discovered substructures are rated by a group of 8 domain experts on a scale of 1 to 5, where 1 means not useful in the domain and 5 means very useful. The number of instances of the discovered substructure that exist in the input database is also listed.

6.1 Evaluation of substructures in a CAD circuit domain

As a result of increased complexity of design and changes in the technologies of implementation of integrated electronic circuitry, the discovery of familiar structures in circuitry can help a designer to understand the design, and to identify common reusable parts in circuitry.

We evaluate SUBDUE by using CAD circuit data representing a sixth-order bandpass "leapfrog" ladder. The circuit is made up of a chain of somewhat similar structures (see Figure 4). We transform the circuit into a graph representation in which the component units and interconnection between several component units appear as vertices and the current flows appear as edges.

Three types of discovered substructures are evaluated: 1) substructures discovered without using the domain knowledge, 2) substructures discovered using the graph match rules, and 3) substructures discovered using a combination of model knowledge and graph match rules. In this domain, the model hierarchy is built based on com-

Figure 4: Bandpass "leapfrog" : sixth-order.

monalities between circuits' functional structure. For example, basic components of a circuit (e.g., resistor, transistor) are represented by primitive nodes, and functional sub-circuits such as operational amplifier, filter, etc. are represented by non-primitive nodes. Furthermore, graph match rules are used to allow two similar components with different labels to be matched.

The performance of the system in this domain is measured in terms of the compression, the computational complexity, and the average human rating. The number of instances of the discovered substructure that exist in the input database is also listed.

The description length of the circuit shown in Figure 4 is 3139.05 bits. Figure 5 shows the substructures discovered in the circuit. The compression value is accumulated from previous iterations. Substructures are labeled with the iteration in which they were discovered. Substructures inside boxes were discovered in previous iterations, and thus form a hierarchy of substructure concepts.

When the model knowledge and graph match rules are used, nine instances of operational amplifier circuits are quickly selected. We also tested SUBDUE's ability to find a hierarchy of substructures. The substructures discovered by SUBDUE for the second iteration represent four instances of inverting integrator circuits which are made up of operational amplifier circuits. For the third iteration, SUBDUE discovered two instances of inverting amplifier circuits which are also made up of operational amplifiers. All of these substructures receive very high human ratings, and represent

Figure 5: CAD circuit–Discovered substructures.

a tremendous reduction in description length. On the other hand, the substructures discovered using the graph match rules offers less compression than the substructures found using no domain knowledge, and both of them receive lower human ratings. Note that the substructure discovered on SUBDUE's first iteration using no domain knowledge receives a high human rating, because the substructure represents an inverter and appears many times in the input graph.

6.2 Evaluation of substructures in an artificial domain

While we have shown the result of evaluations in a real-world domain, we now examine whether such domain knowledge is useful in general. We would like to evaluate whether the domain knowledge can improve SUBDUE's average case performance in an artificially-controlled graph. To test this performance, an artificial substructure is created and is embedded in larger graphs of varying sizes. The graphs vary in terms of graph size and the amount of deviation in the substructure's instances, but are constant with respect to the percentage of the graph that is covered by the substructure's instances. For each deviation value, we run SUBDUE on the graphs until no more compression can be achieved with the following four cases: a) no domain knowledge, b) graph match rules, c) combined model knowledge and graph match rules, and d) combination of a & c. We then measure the compression, the number of nodes expanded, and the number of embedded instances found for all iterations. The effects of the varying deviation values are measured against the average compression value of the four cases mentioned above (Figure 6), the average number of embedded instances found (Figure 7), and the average number of nodes expanded (Figure 8). As the amount of deviation increases, the compression in all four cases decreases as expected. Although case a demonstrates slightly better compression than case c, it is not capable of finding specific relevant substructures. On the other hand, case c demonstrates the least compression, and is capable of finding the embedded substructure. Case b yields the highest compression, but it does not always find the embedded substructure. Case d performs well in both compression and finding the embedded substructures. Hence, the combination of discovery with and without

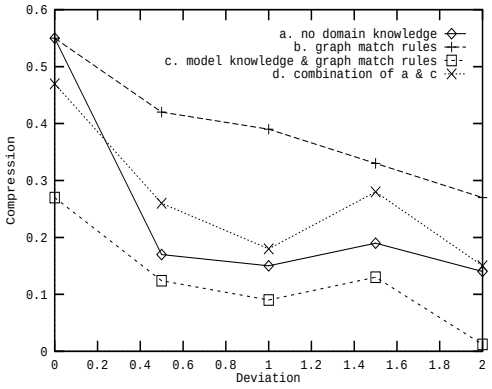

Figure 6: Compression vs. deviation

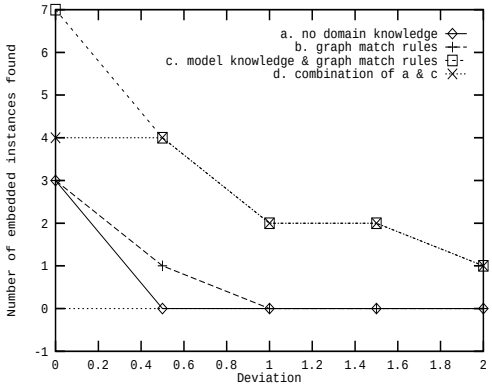

Figure 7: #Instances vs. deviation

domain knowledge performs best as the amount of deviation is increased.

Figure 8 shows that as deviation increases, the run time for case c remains about the same, because the same substructures (of the same size) are found consistently. However, since case d combines both case a and c, and finds varied sizes of substructures, it expands the greatest number of nodes. Because case a and b discover smaller substructures as deviation is increased, they expand fewer nodes.

We again embedded an artificial substructure into larger graphs of varying sizes. Each of the graphs varies in size, as well as in the amount of substructure coverage. For each coverage value, we test the same four cases. The effect of the varying coverage values are measured against the average number of embedded instances found (Figure 9). As coverage increases, cases c and d find an increasing number of embedded instances. Case b finds only a slightly increasing number of instances. On the other hand, case a does not find any instances.

In addition to the domains described here, SUBDUE has been used successfully with and without domain knowledge in a number of other domains including databases of program source code, NASA satellite images, Chinese character databases, and the Brookhaven protein databases [4].

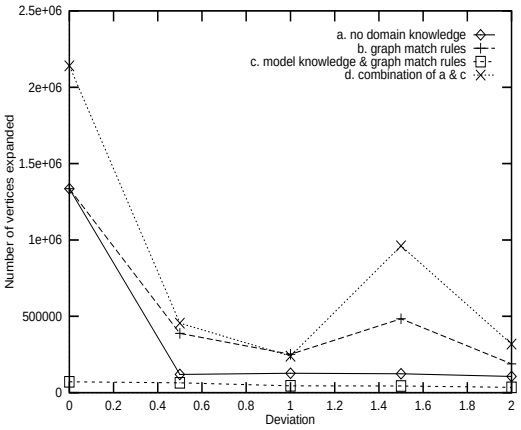

Figure 8: #Expansions vs. deviation

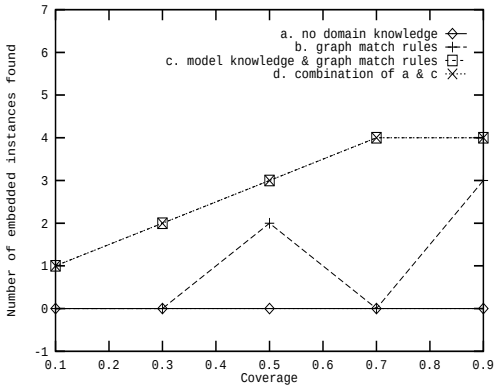

Figure 9: #Instances vs. coverage

7 Scalability

A topic of critical importance in data mining is the scalability of the developed algorithms. In this section we address the issue of scalability of the SUBDUE system. First, we derive an upper bound on the theoretical complexity of the system, and demonstrate how exponential run time performance can be avoided. Second, we present the results of empirical tests on an artificial domain and CAD circuit domains, indicating the run time of SUBDUE as the input graph size increases.

7.1 Theoretical computational complexity analysis

Since algorithms of knowledge discovery in databases always deal with large databases, the issue of computational complexity is very significant. The algorithms employed by SUBDUE are computationally expensive. For example, an unconstrained graph match is exponential in the number of graph vertices. In practice, SUBDUE employs constraints that make the program more scalable. Since the algorithm spends most of its time perform graph matching, the total running time of the algorithm can be expressed as the number of search nodes expanded during graph matches throughout the entire discovery process. In this section, we generate an upper bound on the complexity of SUBDUE as a function of the number of vertices in the input graph.

Additionally, the algorithm without using domain knowledge and the algorithm using domain knowledge are compared.

In what follows, we will be using the following definitions:

- L = user-defined limit on the number of substructures considered for expansion
- nv = number of vertices in the input graph
- $nsub$ = total number of substructures that can be generated
- gm = user-defined maximum number of partial mappings considered during each graph match
- n_{inst} = total number of instances of a given substructure
- m = maximum number of vertices of a model in the model knowledge
- M = average branching factor of a model in the model knowledge
- MC = average number of models that are parent to another model in the model hierarchy
- $N1$ = total number of vertices expanded in SUBDUE without using domain knowledge
- $N2$ = total number of vertices expanded in SUBDUE using model knowledge and graph match rules

7.1.1 Complexity without domain knowledge

This section provides an expression for the computational complexity of the algorithm without using domain knowledge, showing that it depends on the number of vertices in the input graph and the limitations set by the user.

Since the algorithm spends most of its time perform graph matching, the total running time of the algorithm can be expressed as $N1 = nsub \times n_{inst} \times gm$. Considering an upper bound time complexity, assume the input graph is a fully connected graph, where the number of neighbors for a given vertex is $(nv - 1)$, the maximum size of a substructure generated in iteration i of the algorithm is i vertices, and the number of vertices which have already been considered in previous iterations is $(i - 1)$. Hence, the total number of vertices that can be expanded is $((nv - 1) - (i - 1))$. Therefore, the total number of substructures that can be generated is $nsub = \sum_{i=1}^{L} i \times ((nv -$

$1) - (i - 1))$.

The total number of instances needed to be compared for a given substructure involves the instances of the substructure itself and the instances of the substructure's parent. For a substructure with i vertices, the maximum number of non-overlapping instances is $\frac{nv}{i}$. Since we consider an upper bound case, let the maximum number of non-overlapping instances be nv. Hence, the total number of instances needed to be compared for a given substructure is $n_{inst} = nv \times (L - 1)$.

We have shown that by placing a limit on gm and L, the time complexity for the graph matching is polynomial in nv. If either of the two limits L or gm are removed, the complexity of the discovery algorithm becomes exponential in nv. We are currently developing a parallel implementation of SUBDUE that may further improve the scalability of the algorithm.

7.1.2 Complexity using domain knowledge

This section provides an expression for the computational complexity of the algorithm using domain knowledge, showing that it depends on the number of vertices in the input graph, the limitation set by the user, and the model knowledge used. We will point out that for the upper bound case, the number of vertices expanded for discovery using domain knowledge can be less than the number of vertices expanded for discovery without using domain knowledge under certain circumstances.

Since the algorithm searches not only for the instances of a substructure, but also for a model in the model hierarchy which matches the substructure, the total running time of the algorithm can be expressed as $N2 = (nsub \times n_{inst} \times gm) + (nsub \times M \times MC \times gm)$, where the first term is the number of vertices expanded for the search of the substructures' instances, and the second term is the number of vertices expanded during the search for a model in the model hierarchy.

The maximum number of expanded nodes for a substructure is limited to the maximum number of vertices of a model in the model hierarchy (m). Hence, the number of iterations is limited to m. Therefore, $nsub = \sum_{i=1}^{m} i \times ((nv - 1) - (i - 1))$. The total number of instances needed to be compared for a given substructure is

$n_{inst} = nv \times (m - 1)$.

We have shown that by placing a limit on gm, the time complexity for the graph match algorithm is polynomial in nv. If the gm limitation is removed, the complexity of the discovery algorithm becomes exponential in nv.

$(M \times MC)$ is dependent upon the size of the model knowledge. In general, L is set to half the size of the input graph, gm is set to the fourth power of the size of a substructure or model, whichever is bigger. Therefore, L is much larger than m. When the size of a substructure is big, which means that $(M \times MC)$ is small compared to gm, and $(M \times MC)$ is negligible, the number of nodes expanded for discovery using domain knowledge is less than the number of nodes expanded for discovery without domain knowledge.

In conclusion, the number of nodes expanded for discovery using domain knowledge and without domain knowledge depends on the size of the input graph and model knowledge (m, M, MC), the size of the discovered substructures, and the limitations set by the user.

7.2 Empirical Scalability Results

The previous section derived upper bounds on SUBDUE's run-time performance. Here we empirically demonstrate SUBDUE's run-time performance on increasing database sizes. Figure 10 shows the results of the first experiment, which measures SUBDUE's run time when used to discover substructures embedded in artificial graphs. The x axis of the graph represents the size of the artificial input graph in number of vertices. The graphs are designed to contain twice as many edges as vertices. For each graph size, six random graphs are generated. The graphs vary in terms of the type of substructure embedded, the number of instances of the substructure, and the amount of deviation in the substructure instances.

As can be seen from the graph in Figure 10, SUBDUE's run time increases polynomially with the size of the input graph. Figure 11 demonstrates similar results when SUBDUE is applied to increasing sizes of CAD graphs representing portions of an A-to-D converter. For this experiment, a CAD circuit database was provided

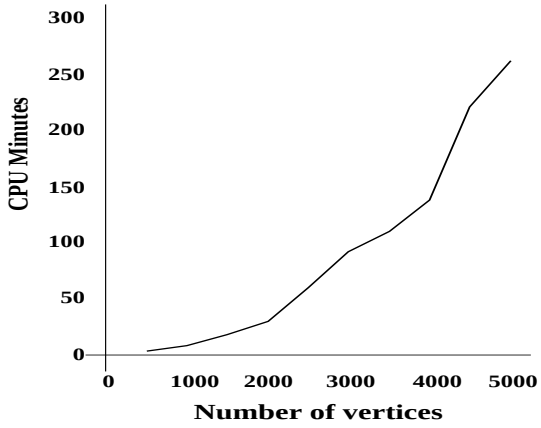

Figure 10: Artificial graph scalability results.

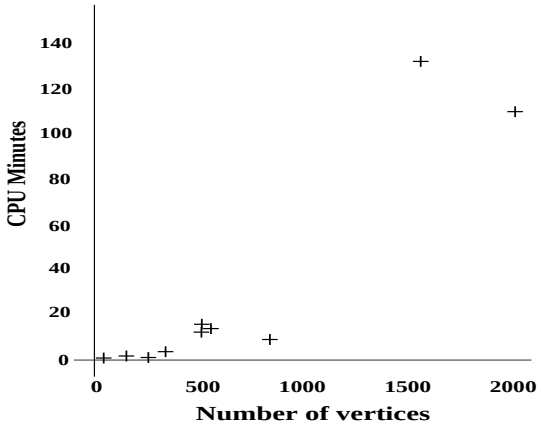

Figure 11: CAD scalability results.

by National Semiconductor. The original circuit was described in terms of modular sub-circuit units. We flattened the database, removing the sub-circuit definitions, to let SUBDUE rediscover the modular components. The graphs in this experiment represent sub-circuits from the original circuit database. Although variations in graph features cause variations in run time, Figure 11 again indicates a polynomial increase in run time as the graph size increases linearly. To provide a basis of comparison, the CAD circuit shown in Figure 4 is represented using a graph with 156 vertices and 200 edges.

While the theoretical complexity indicates exponential run time of the system when the graph match and number of substructure definitions is unconstrained, these empirical results demonstrate a polynomial increase in run time. There are several reasons for the improvement in actual performance over the theoretical upper bound. First, the graphs used for the experiments in this section have a low amount of connectivity. As the ratio of edges to vertices in the input graph increases, both the potential number of substructures and the amount of work inherent in graph match increases. To demonstrate the effect on performance, Figure 12 graphs the run time of SUBDUE as the number of vertices in the input graph remains at 200. Connectivity here reflects the ratio of number of edges to number of vertices. In practice, a low

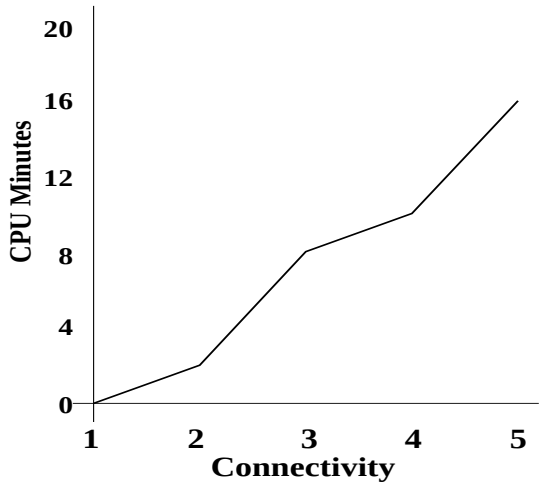

Figure 12: Effect of connectivity on performance.

amount of connectivity is usually found. Of the over 100 graphs tested in a variety of domains, the average ratio of edges to vertices is 1.8.

Although connectivity yields the most dramatic effect on performance, other features of the input graph can also affect SUBDUE's run time. These features include the number of unique vertex labels, unique edge labels, the size of the best discovered substructure, and variance in instances of discovered substructures. As mentioned in the previous subsection, SUBDUE's performance can be guided manually to optimize the tradeoff between value of discovered substructures and run time. In particular, by placing a limit on gm, the complexity of each graph match can be limited at the expense of a less-accurate determination of similarity between substructure instances. Similarly, the value of L can be changed to limit the number of substructures that are considered for expansion. These parameters can be used to ensure polynomial run-time performance with even the most complex input graphs.

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 knowledge in structural databases.

This paper describes 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, computer programs, image data, chemical compound data, etc. This integration improves SUBDUE's ability to both compress an input graph and discover substructures relevant to the domain of study. We also analyzed the complexity of the algorithms and showed that by placing computational constraints on the graph match and substructure expansion, exponential behavior of the algorithm can be avoided. These complexity results combined with the results of experiments on databases of increasing size verify the scalability of the SUBDUE system.

We are continuing to demonstrate SUBDUE's behavior on a wide range of discovery tasks. Because scalability is of paramount importance for data mining applications, we are also currently developing a parallel MIMD version of SUBDUE and a version that runs on a distributed network of workstations.

References

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

[2] D. Conklin. Machine discovery of protein motifs. *Machine Learning*, 21:125–150, 1995.

[3] 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.

[4] S. Djoko. *The role of domain knowledge in substructure discovery.* PhD thesis, Department of Computer Science and Engineering, University of Texas at Arlington, 1995.

[5] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, 1996.

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

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

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

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

[10] 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.* Morgan Kaufmann Publishers, Inc., 1991.

Diane Cook is currently an Assistant Professor in the Computer Science and Engineering Department at the University of Texas at Arlington. Her research interests include artificial intelligence, machine planning, machine learning, robotics, and parallel algorithms for artificial intelligence. Dr. Cook received her B.S. from Wheaton College in 1985, and her M.S. and Ph.D. from the University of Illinois in 1987 and 1990, respectively. Her current home page address is http://www-cse.uta.edu/~cook/home.html.

Lawrence Holder is currently an Assistant Professor in the Department of Computer Science and Engineering at the University of Texas at Arlington. His research interests include artificial intelligence and machine learning. Dr. Holder received his M.S. and Ph.D. degrees in Computer Science from the University of Illinois at Urbana-Champaign in 1988 and 1991. He received his B.S. degree in Computer Engineering also from the University of Illinois at Urbana-Champaign in 1986. His current home page address is http://www-cse.uta.edu/~holder/home.html.

Surnjani Djoko is currently a Member of Scientific Staff at Bell Northern Research, Richardson, TX. Her research interests are in the areas of knowledge discovery in databases, machine learning, statistical methods for inducing models from data, and parallel algorithms. Dr. Djoko received the B.S.E.E. degree from Tamkang University, Taiwan, R.O.C. in 1986, the M.S.E.E. degree and the Ph.D. degree in Computer Science and Engineering from the University of Texas at Arlington, TX, in 1989 and 1995, respectively.