Analyzing the Benefits of Domain Knowledge in Substructure Discovery *

Surnjani Djoko, Diane J. Cook and Lawrence B. Holder
University of Texas at Arlington
Department of Computer Science and Engineering
Box 19015, Arlington, TX 76019
djoko@cse.uta.edu, cook@cse.uta.edu, holder@cse.uta.edu

Abstract

Discovering repetitive, interesting, and functional substructures in a structural database improves the ability to interpret and compress the data. However, scientists working with a database in their area of expertise often search for a predetermined type of structure, or for structures exhibiting characteristics specific to the domain. This paper presents methods 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. The domain knowledge is incorporated into SUBDUE following a single general methodology to guide the discovery process. Results show using domain-specific knowledge improves the search for substructures which are useful to the domain, and leads to greater compression of the data. To illustrate these benefits, examples and experiments from the domain of computer programming, computer aided design circuit, and a series of artificially-generated domains are presented.

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

1 Introduction

With the increasing amount and complexity of today's data, there is an urgent need to accelerate the discovery of information and the generation of knowledge from large databases. To date, the SUBDUE system has been used to discover interesting and repetitive substructures in structural data [CH94]. The substructures are evaluated both by a set of domain independent heuristics and by the substructures' ability to describe and compress the original data set based on the minimum description length (MDL) principle [Ris89]. Once the

*Supported by NASA grant NAS5-32337.

substructures are discovered, they are used to simplify the data by replacing instances of the substructure with a pointer to the substructure definition. The discovered substructures allow abstraction over detailed structure in the original data. Iteration of the substructure discovery and replacement process constructs a hierarchical description of the structured data in terms of the discovered substructures. This hierarchy provides varying levels of interpretation that can be accessed based on the goals of the data analysis.

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 the discovery process. Domain independent heuristics and discovery techniques are 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.

This paper focuses on methods 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 to discover substructures, and interestingness of the discovered substructures. These methods are generally applicable to most structural data, such as computer-aided design (CAD) circuit data, computer programs, chemical compound data, and image data.

2 Adding domain knowledge to the SUBDUE system

We now present several types of domain knowledge that are used in the discovery process, explain how they bias discovery of certain types of substructures, and detail how the knowledge is added to the SUBDUE system. Although the minimum description length principle still drives the discovery process, domain knowledge is used to input a bias toward certain types of substructures. The intuition behind this approach is that experts often have a preference for particular types of discoveries. Domain knowledge can be used to isolate those aspects of substructures they do understand, and to help constrain the discovery process.

2.1 Model/Structure knowledge

Model/Structure knowledge provides to the discovery system specific types of structures that are likely to exist in the database and are of particular interest to a scientist using the system. The input structures are organized in a hierarchy. Leaves in the hierarchy represent primitive (nondecomposable) structures which are basic elements of the domain, and inner nodes represent nonprimitive structures. Nonprimitive structures consist of a conglomeration of primitive vertices and/or lower-level nonprimitive vertices. 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.

In the programming domain, for example, special symbols are represented by primitive vertices, and functional subroutines (e.g., swap, sort, increment) are represented by non-

primitive vertices. In the CAD circuit domain, the basic components of a circuit (e.g., resistor, transistor) are represented by primitive vertices, and functional subcircuits such as operational amplifier, filter, etc. are represented by non primitive vertices. This indexed hierarchical representation allows examining of the structural knowledge at various level of abstraction, focusing the search and reducing the search space.

2.1.1 Using structure knowledge to guide the discovery

The modified version of SUBDUE can be biased to look for structures of the type specified in the model/structure hierarchy. The model/structure pointed to by the matched model vertex is selected as a candidate model to be matched with the input substructure. Each iteration through the process selects a substructure from the input graph which has the best match to the selected model according to its ability to compress the entire input graph. Selected substructures are incrementally expanded as possible candidates for the next iteration. The process searches for the best substructure until all possible substructures have been considered or until a substructure has been found that matches the selected model.

The compressed graph is encoded as described elsewhere [CH94]. After a substructure is discovered, each instance of the discovered substructure is encoded as an index to the corresponding substructure or model definition. The discovered substructure is represented using $I(S)$ bits which is the number of bits needed to encode the model index, and the graph after the substructure replacement is represented in $I(G|S)$ bits. SUBDUE searches for the substructure S in graph G minimizing $I(S) + I(G|S)$.

2.1.2 Combining substructure discovery with and without model knowledge

In order not to overly bias the discovery process toward certain types of substructure based on the model/structure knowledge, the discovery process can combine the discovery without using model/structure knowledge with the discovery process using structure knowledge. Using domain-independent and domain-dependent knowledge together may be more useful than using either type of approach in isolation. In particular, using domain-dependent knowledge alone may block the discovery of unexpected substructures. However, domain-dependent knowledge can assist the discovery process by focusing search and ensuring that the discovered substructure is meaningful to the user.

In each iteration of the algorithm, the SUBDUE system discovers at most two best substructures, one based on no model knowledge and the other based on model knowledge. These substructures are used to compress the input graph. SUBDUE selects the compressed which requires the smallest description length as the input graph for the next iteration of the discovery process. The compressed graph which has not been selected is added to the list of unprocessed graphs. If after further iterations, SUBDUE obtains a compressed graph whose amount of compression is smaller than any compressed graph in the list of unprocessed graphs, this compressed graph is put on the list of unprocessed graphs. SUBDUE resumes the discovery process using the compressed graph from the list of unprocessed graphs which has the maximum amount of compression. This process is repeated until the list of unprocessed graphs is exhausted. The MDL principle is used as a compression measure for both using the model knowledge and without using the model-based discovery.

2.2 Graph match rules

At the heart of the SUBDUE system lies an inexact graph match algorithm that finds instances of a substructure definition. The graph match is used to identify isomorphic substructures in the input graph. Many of those substructures could show up in a slightly different form throughout the data. These differences may be due to noise and distortion, or may illustrate slight differences between instances of the same general class of structure. Each distortion of a graph is assigned a cost. A distortion is described in terms of basic transformations performed by the graph match such as deletion, insertion and substitution of vertices and edges. Given $g1$ and $g2$, and a set of distortion costs, the actual computation of $matchcost(g1, g2)$ can be performed using a tree search procedure. As long as $matchcost(g1, g2)$ does not exceed the threshold set by the user, the two graphs $g1$ and $g2$ are considered to be isomorphic.

By using graph match rules, each transformation is assigned a cost based on the domain of usage. Consider an example in the programming domain. We allow a vertex representing a variable to be substituted by another variable vertex, and do not allow a vertex representing an operator which is a special symbol, a reserved word, or a function call, to be substituted by another vertex. These rules can then be represented as the following:

IF (programming domain) and (substitute variable vertex) THEN graph match cost = 0.0;
IF (domain = programming) and (substitute operator vertex) THEN graph match cost = 2.0;

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 domain graph.

2.3 Feature knowledge

Domain-specific rules can be used to generate new features describing the data. Because we know that different domains will have features not explicitly represented in the original database which can be extracted to provide more understanding toward the domain, we incorporate domain feature knowledge into the system to automatically generate these domain features. The generated features should be considered of great value in understanding the domain. Feature knowledge captures the relations among the substructures in the domain, generates important features of the input graph to provide helpful information, and provides a step towards understanding of the input graph.

Consider an example in the programming domain. Feature knowledge specifies how to generate a loop feature whenever repetitive subcodes/substructures appear consecutively in a program and replace the substructures with the loop structure, rendering the program's meaning clearer; to perform substitution of variable definitions, enabling the system to extract formulae. The results appear to be an effective aid to graph understanding. Feature generation is supplied as a preprocessing step to the discovery system.

3 Evaluation of knowledge in SUBDUE's discovery

In this section, we evaluate the benefits and costs of utilizing domain specific information in performing substructure discovery. We will measure the performance of SUBDUE with and without domain-specific information when applied to databases in the programming,

```
sorted = 0;                                          /* bubble sort */
while(sorted == 0)
    sorted = 1;
    for(j = 0; j < listsize − 1; j++)
        if(list[j] > list[j + 1])
            temp = list[j]; list[j] = list[j + 1]; list[j + 1] = temp; sorted = 0;
for(gap = n/2; gap > 0; gap = gap/2)                          /* shell sort */
    for(i = gap; i < n; i++)
        for(j = i − gap; j >= 0&&v[j] > v[j + gap]; j = j − gap)
            temp = v[j]; v[j] = v[j + gap]; v[j + gap] = temp;
for(i = n; i > 0; i − −)      /* bubble sort operates here as a type of selection sort */
    for(j = 2; j >= i; j++)
        if(a[j − 1] > a[j])
            t = a[j − 1]; a[j − 1] = a[j]; a[j] = t;
```

Figure 1: A sample program of three different sort procedures.

computer aided design and artificial domains. The goals of our substructure discovery system are to efficiently find substructures that can reduce the amount of information needed to describe the data, and to find substructures that are considered interesting and useful for the given domain.

To evaluate SUBDUE in a programming and CAD circuit domain, we compare SUBDUE's discovered substructures to human ratings. If the domain-dependent approach has some validity, SUBDUE should prefer the substructures which were rated higher by humans.

The discovered substructures are evaluated in three ways, 1) without using the domain knowledge, 2) using the graph match rules and 3) using a combination of model/structure knowledge and graph match rules. The performance of the system is measured along three dimensions: 1) the total description length[1], 2) the number of search nodes expanded by SUBDUE before the substructures are discovered, and 3) the interestingness of the 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 least preferred and 5 means most preferred. We then compute the average and the standard deviation of the ratings of each discovered substructure. The number of instances in the database of the discovered substructure is also listed.

3.1 Evaluation of the substructures in the programming domain

The discovery of familiar structures in a program can help a programmer to understand the function and modularity of the code. Automating this discovery process will facilitate many tasks that require program understanding, e.g., maintenance, translation, and debugging. In order to determine the value of substructures discovered by SUBDUE, we concatenate three different sort routines into one program (see Figure 1), and transform it into a graph representation which is independent of the source language.

[1]Total description length is the description length of the discovered substructure plus the description length of the compressed input graph using the discovered substructure.

Table 1: Program – Discovered substructures.

The description length of the sample program in Figure 1 is 2598.99 (in bits). Table 1 shows three discovered substructures of the sample program with their total description length, number of nodes expanded for the graph match, number of instances found, and the average and standard deviation of the human rating. The substructure with no domain knowledge has the highest total description length, and the lowest human rating. Although the substructure found using the graph match rules alone has the lowest total description length, it does not yield a good human rating. On the other hand, the "simple swap" substructure found using both model knowledge and graph match rules has a total description length lower than the substructure with no domain knowledge and received the highest human rating.

3.2 Evaluation of the substructures in the 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 complex circuitry can help a designer to understand the layout, and to identify common reusable parts in the circuitry.

We evaluate SUBDUE by using CAD circuit data representing a sixth-order bandpass "leapfrog" ladder [Bru80]. The circuit is made up of a chain of somewhat similar structures (see Figure 2). We transform the circuit into a graph representation in which the component units appear as vertices and the current flows appear as edges. The description length of the circuit in Figure 2 is 3139.05 (in bits). The numbers in circles shown in Figure 3 represent the iteration in which the substructure is discovered.

Figure 2: Bandpass "leapfrog" : sixth-order.

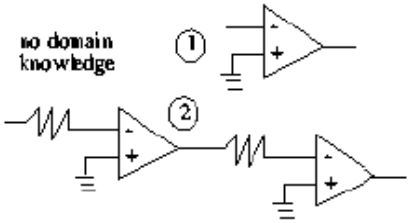
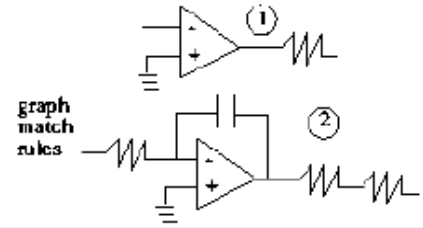
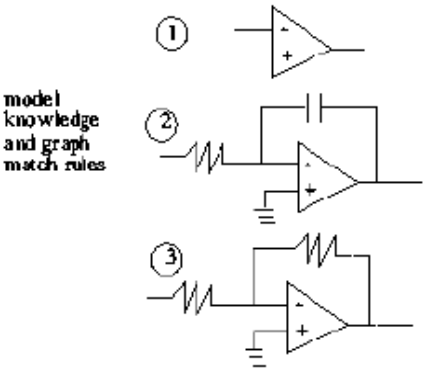
Usage of Domain Knowledge	Discovered Substructures	Total Description Length (in bits)	Number of Nodes Expanded	Human Rating Average [Standard Deviation]	Number of Instances
no domain knowledge		1170.18	571370	3.8 [1.3]	9
		997.4	46422	2.7 [1.1]	3
graph match rules		1781.34	145175	2.7 [1.6]	6
		1523.69	39881	2.7 [0.9]	3
model knowledge and graph match rules		1472.67	24273	4 [1.4]	9
		1070.83	12960	4.2 [1.2]	4
		878.1	7432	4.2 [1.2]	2

Figure 3: CAD circuit – Discovered substructures.

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 rating, and have a tremendous reduction in the amount of total description length. On the other hand, the substructures with graph match rules has lesser compression than the substructures with no domain knowledge, and both of them receive low human rating.

The result also shows that for about the same size of substructures, the nodes expanded for discovery with the domain knowledge is lesser than the nodes expanded for discovery with no domain knowledge.

3.3 Evaluation of the substructures in the artificial domain

While we have shown results of evaluations in two domains, we now examine whether such domain knowledge is useful in general. We would like to evaluate whether domain knowledge can improve SUBDUE's average case performance in artificially-controlled graphs. To test this performance, an artificial substructure is created and is embedded in larger graphs of three varying sizes. The graphs vary in terms of graph size and amount of deviation in the substructure instances, but are constant with respect to the percentage of the graph that is covered by substructure instances. For each deviation value, we run each of 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 the all iterations. The effects of the varying deviation values are measured against the average compression value of the four cases mentioned above (Figure 4) and the average number of nodes expanded (Figure 4). As the deviation is increased, the compression of all four cases decreases as expected. Although case a has slightly better compression than case c, it is not capable of finding specific relevant substructures. On the other hand, case c has the least compression, and is capable of finding the embedded substructure. Case b has the highest compression, but it does not perform well for finding the embedded substructure. The last case is case d, which performs well in both compression and finding the embedded substructures. Hence, the combination of discovery with and without domain knowledge performs the best as the deviation is increased.

Figure 4 shows that as the deviation is increased, the number of nodes expanded for case c remain 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 varies sizes of substructures, it expands the most number of nodes. As case a and b discover smaller substructures as the deviation is increased, they expand lesser number of nodes.

Second, we again embedded an artificial substructure into larger graphs of three varying sizes. Each of the graphs varies in the size, and the amount of the input graph covered by the embedded substructure. 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

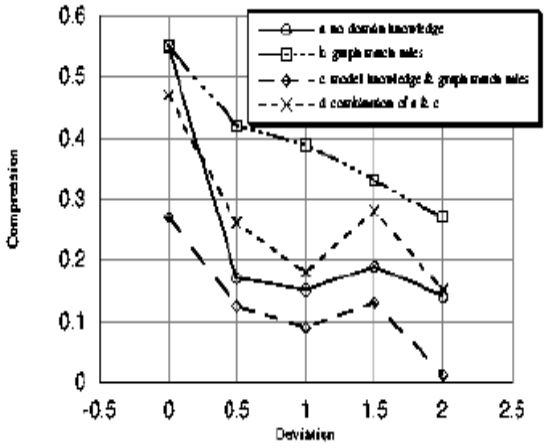


Figure 4: Affect of deviation on compression and number of nodes expanded.

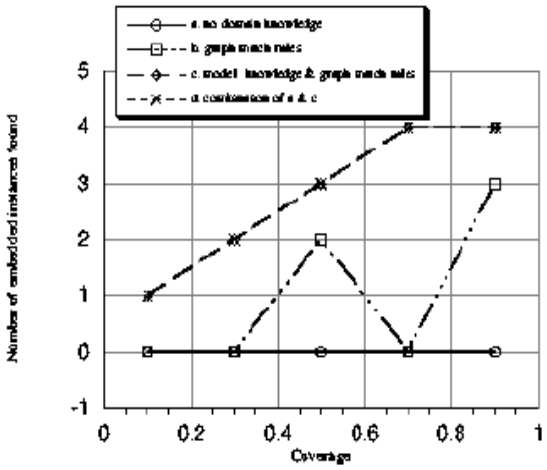


Figure 5: Coverage versus number of instances found.

found (Figure 5). As the coverage is increased, cases c and d find increasing number of embedded instances. Case b find only slightly increasing number of instances. On the other hand, case a does not find any instances.

4 Conclusions

SUBDUE is a system devised for experimenting with automated discovery by using domain knowledge. This approach requires that the domain knowledge be as generic as possible and can be reused over a class of similar applications.

Few current discovery systems integrate domain knowledge seamlessly into the discovery process. This paper describes methods for integrating domain independent and domain dependent substructure discovery based on the minimum description length principle. These

methods are generally applicable to most structural data, such as computer aided design (CAD) circuit data, computer programs, chemical compound data, image data. This integration improves SUBDUE's ability to both compress the input graph and discover substructures relevant to the domain of study. The result also shows that the number of nodes expanded in each iteration of the graph match procedure depends on the amount of domain knowledge usage and the size of the substructure found.

Our future work will evaluate the benefits of domain knowledge applied to NASA image data, where the domain knowledge can be obtained from experts in the field of the analysis. Furthermore, since many rules about a domain are incomplete, and uncertainty can arise because of incompleteness and incorrectness in the domain expert's understanding of the properties of the environment, the inclusion of uncertain knowledge will be pursued.

References

[Bru80] L. T. Bruton. *RC-active circuits.* Prentice-Hall, 1980.

[CH94] 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.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Company, 1989.