

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 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. The domain knowledge is incorporated into SUBDUE following a single general methodology 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. To illustrate these benefits, examples and experiments from the computer programming, computer aided design circuit, and artificially-generated domains are presented.

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 (Holder, Cook, & Djoko 1994). 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 (Quinlan & Rivest 1989). Once the 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 structural data in

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

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 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 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 many structural data, such as computer-aided design (CAD) circuit data, computer programs, chemical compound data, and image data.

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 toward 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

*Supported by NASA grant NAS5-32337.

Model/Structure knowledge

Model/Structure knowledge provides to the discovery system specific types of structures that are likely to exist in the database and that 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 nonprimitive 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 nonprimitive vertices. This indexed hierarchical representation allows examining of the structural knowledge at various level of abstraction, focusing the search and reducing the search space.

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 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 either a substructure has been found that matches the selected model 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. If *compression* is greater than zero, the representation of G using S which match the model M is used instead of the default representation (*compression* is defined as $(1 - \frac{DL \text{ of compressed graph}}{DL \text{ of input graph}})$), where DL is description length).

The compressed graph is encoded as described elsewhere (Holder, Cook, & Djoko 1994). After a substructure is discovered, each instance of the discovered substructure is encoded as an index to the correspond-

Combining substructure discovery with and without model knowledge. In order not to overly bias the discovery process toward certain types of substructures based on the model knowledge, the discovery process can combine the discovery without using model knowledge with the discovery process using model 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 discovered using only the MDL principle (without domain knowledge), and the other discovered using the MDL principle and model knowledge (with domain knowledge). Each of the substructures is used to compress the input graph. SUBDUE selects the compressed graph yielding the maximum amount of compression as the input graph for the next iteration of the discovery process. The compressed graph which has not been selected is put in the unprocessed list. If after further iterations, SUBDUE obtains a compressed graph whose amount of compression is smaller than any compressed graph in the unprocessed list, this compressed graph is not processed immediately, but is inserted into the unprocessed list according to its *compression* value. SUBDUE resumes the discovery process using the compressed graph from the unprocessed list which has the maximum amount of compression. This discovery is repeated until the unprocessed list is exhausted. The MDL principle is used as a compression measure for both using the model knowledge and without using the model-based discovery.

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 g_1 and g_2 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;

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.

Complexity analysis. 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 makes use of constraints that makes the program more scalable. In this section, we will generate an upper bound on the complexity of SUBDUE as a function of the number of nodes in the input graph.

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

n = the number of a substructure's instances found in the input graph,

L = the user-defined limit on the number of substructures considered for expansion,

nv = the number of nodes in the input graph,

$nsub$ = the total number of substructures that can be generated,

gm = the user-defined maximum number of partial mappings that are considered during each graph match,

nn = the total number of instances needed to be compared for a given substructure, and

B = the total number of nodes expanded.

Since the algorithm spends most of its time perform graph matching, the total running time of the algorithm can be expressed as

$$B = nsub \times nn \times gm$$

Considering an upper bound time complexity, assume the input graph is a fully connected graph, where the number of neighbors for a given node is $(nv - 1)$, the maximum size of a substructure generated in iteration i of the algorithm is i nodes, and the number of nodes which has already been considered in previous iteration of i is $(i - 1)$. Hence, the total number of possible expansion node 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 nodes, the maximum number of nonoverlapping instances is $\frac{nv}{i}$. Since we consider an upper bound case, the maximum number of nonoverlapping instances is nv . Hence, the total number of instances needed to be compared for a given substructure is

$$nn = nv \times (L - 1)$$

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 become exponential in nv . We are currently developing a parallel implementation of SUBDUE that may further improve the scalability of the algorithm.

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 statements/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.

Evaluation of SUBDUE's domain-independent versus domain-dependent discovery

In this section, we evaluate the benefits 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, circuit, and artificially-generated 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.

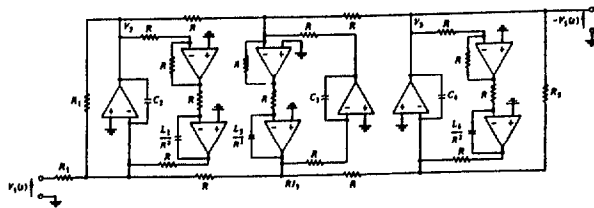


Figure 2: Bandpass "leapfrog" : sixth-order.

Usage of Domain Knowledge	Discovered Substructures	Compression	Number of Nodes Expanded	Human Rating Average [Standard Deviation]
no domain knowledge	①	0.63	571370	4.2 [1.2]
	②	0.68	46422	2.7 [1.2]
graph match rules	①	0.43	145175	2.7 [1.7]
	②	0.51	39881	2.7 [1.0]
model knowledge & graph match rules	①	0.53	24273	4.3 [1.2]
	②	0.66	12960	4.5 [0.8]
	③	0.72	7432	4.5 [0.8]

Table 2: Circuit - Discovered substructures.

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 (Bruton 1980). 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 Table 2 represent the iteration in which the substructure is discovered.

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 represent a tremendous reduction in description length. On the other hand, the substructures found using the graph match rules (with 6 instances) offers lesser compression than the substructures found using no domain knowledge (with 9 instances), and both of them receive low human rating.

The result again shows that the time complexity of SUBDUE depends on the amount of domain knowledge used and the size of the substructure found. Furthermore, SUBDUE's domain dependent discovery process perform better of finding relevant substructures than domain independent.

Evaluation of substructures in artificially-generated domain

While we have shown the result of evaluations in two domains, 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 3) and the average number of nodes expanded (Figure 4). 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 has the highest compression, but it does not perform well of 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 best as the amount of deviation is increased.

Figure 4 shows that as deviation is increased, the number of nodes expanded 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

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.

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.

Evaluation of substructures in programming domain

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

The description length of the sample program shown in Figure 1 is 2598.99 (in bits). Table 1 shows the first iteration of three discovered substructures of the sample program with: 1) compression, 2) time complexity, and 3) human rating. The substructure found using no domain knowledge (with 9 instances) has the lowest compression and human rating. Although the substructure found using the graph match rules alone (with 8 instances) has the highest compression, it does not yield a good human rating. On the other hand, the substructure found using both model knowledge and graph match rules (with 3 instances) has a compression higher than the substructure found using no domain knowledge and it receives the highest human rating.

Evaluation of substructures in 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

```

{bubble sort}
sorted = 0;
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;
{shell sort} .
for (gap = n/2; gap > 0; gap = gap/2)
    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;
{bubble sort operates as a type of selection sort}
for (i = n; i > 0; i--)
    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: Part of a sample program concatenating three different sort procedures.


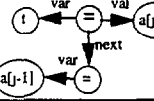
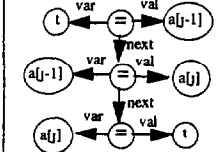
Usage of Domain Knowledge	Discovered Substructures	Compression	Number of Nodes Expanded	Human Rating Average [Standard Deviation]
no domain knowledge		0.07	68386	1.33[1.23]
graph match rules		0.22	93440	2.0[1.41]
model knowledge & graph match rules		0.11	122714	4.83[0.41]

Table 1: Program - Discovered substructures.

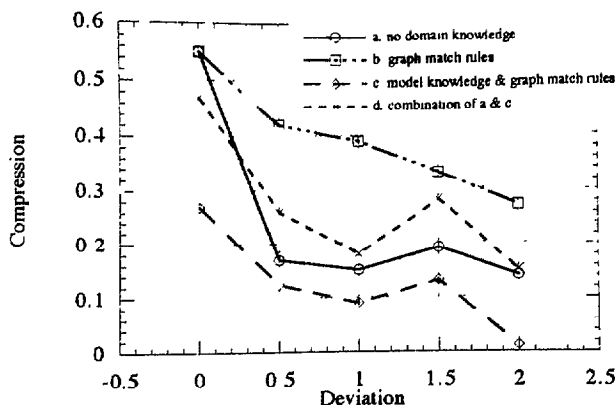


Figure 3: Deviation versus compression.

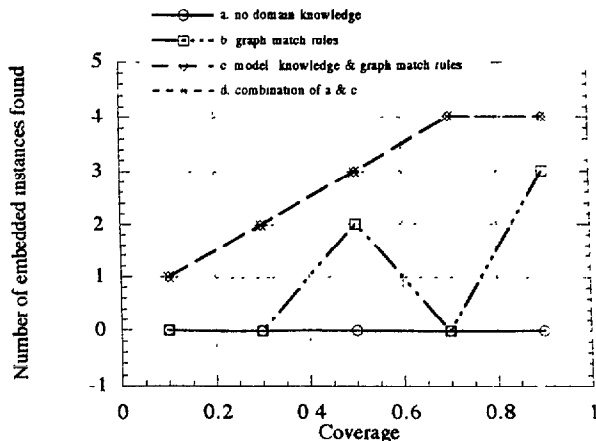


Figure 5: Coverage versus number of instances found.

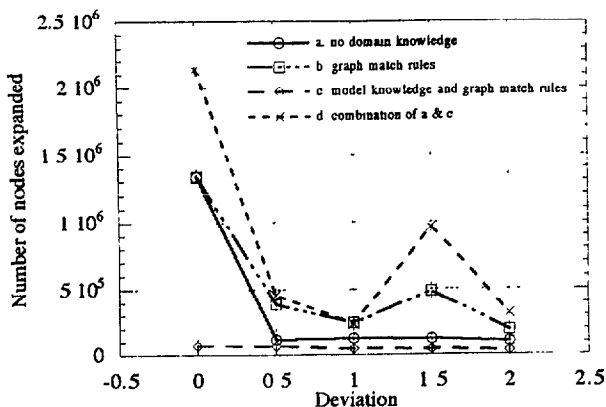


Figure 4: Deviation versus number of nodes expanded.

substructures, it expands the most number of nodes. Because case a and b discover smaller substructures as deviation is increased, they expand lesser number of nodes.

We again embedded an artificial substructure into a larger graphs of varying sizes. Each of the graphs varies in the size, as well as 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 found (Figure 5). As the coverage is increased, cases c and d find an increasing number of embedded instances. Case b finds only slightly increasing number of instances. On the other hand, case a does not find any instances.

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.

This paper describes a method for integrating domain independent and domain dependent substructure discovery based on the minimum description length principle. This method is generally applicable to many structural databases, such as computer aided design (CAD) circuit data, computer programs, 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. The result also shows that the time complexity of the discovery process depends on the amount of domain knowledge used and the size of the substructure found.

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

- Bruton, L. T. 1980. *RC-active circuits theory and design*. Prentice Hall.
- Holder, L. B.; Cook, D. J.; and Djoko, S. 1994. Substructure discovery in the subdue system. In *Proceedings of Knowledge Discovery in Databases Workshop (KDD-94)*, 169-180.
- Quinlan, J. R., and Rivest, R. L. 1989. Inferring decision trees using the minimum description length principle. *Information and Computation* 80:227-248.